LEMBAR PENGESAHAN

METODE *FAILOVER* UNTUK MENGATASI POWER *LOSS* PADA *NODE WIRELESS SENSOR NETWORK* (WSN)

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI REKASAYA KOMPUTER

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Nararya Andika Sujarwo

NIM. 135060300111047

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing
pada tanggal 21 Desember 2018

Mengetahui Ketua Jurusan Teknik Elektro	Menyetujui Dosen Pembimbing
Hadi Suyono, S.T., M.T., Ph.D., IPM.	**Adharul Muttaqin, S.T., M.T.**
NIP. 19730520 200801 1 013	**NIP. 19760121 200501 1 001**

JUDUL SKRIPSI :

METODE *FAILOVER* UNTUK MENGATASI POWER *LOSS* PADA *NODE WIRELESS SENSOR NETWORK* (WSN)

Nama Mahasiswa	: Nararya Andika Sujarwo
NIM	: 135060300111047
Program Studi	: Teknik Elektro
Konsentrasi	: Rekayasa Komputer

KOMISI PEMBIMBING :

Ketua : Adharul Muttaqin, S.T., M.T. .....

TIM DOSEN PENGUJI :

Dosen Penguji 1 : Raden Arief Setiawan, S.T., M.T .....

Dosen Penguji 2 : Dr. Ir. Muhammad Aswin, M.T. .....

Dosen Penguji 3 : Waru Djuriatno, S.T., M.T. .....

Tanggal Ujian : 11 Desember 2018

SK Penguji : 2819/UN10.F07/SK/2018

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Allah SWT, karena atas segala pertolongan dan perlindunganNya skripsi berjudul "Metode *Failover* Untuk Mengatasi *Power Loss* Pada *Wireless Sensor Network* (WSN)" dapat diselesaikan. Penulis menghaturkan rasa terimakasih dan apresiasi yang sebesar-besarnya atas bantuan dalam penyelesaian skripsi ini kepada:

- Keluarga tercinta, ayah (Eko Sujarwo) yang selalu mengingatkan, memberi semangat serta motivasi, ibu (Indriana Maria Josphine Tutuhatunewa) yang selalu mendoakan dan memberi semangat penulis dalam segala kondisi,
- Bapak Adharul Muttaqin, ST., MT. selaku Ketua Kelompok Dosen Keahlian Konsentrasi Rekayasa Komputer serta selaku Dosen Pembimbing, atas segala bimbingan, pengarahan, bantuan, serta kritik dan saran dalam kelancaran studi maupun skripsi,
- Terimakasih buat kakak serta adik yang selalu memberikan motivasi untuk menyelesaikan penulisan skripsi ini,
- Terimakasih calon pendamping hidup Marisa Primardiansya yang selalu menemani dan memberikan dukungan serta do'a,
- Bapak Ir. Hadi Suyono, ST., MT., Ph.D., IPM.selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya,
- Ibu Ir. Nurussa'adah, MT. selaku Sekretaris Jurusan Teknik Teknik Elektro Universitas Brawijaya atas segala motivasi, pengarahan, serta kritik dan saran dalam kelancaran studi maupun skripsi,
- Bapak Ali Mustofa, ST., MT. selaku Ketua Program Studi S1 Teknik Elektro Universitas Brawijaya,
- Seluruh Dosen Pengajar dan Staff Recording Teknik Elektro Universitas Brawijaya,
- Teman- teman Spectrum angkatan 2013,
- Rekan- rekan pejuang skripsi Edy, Nardo Golan,terimakasih atas niat, semangat, dan motivasi, dan bantuan yang diberikan dalam proses pengerjaan skripsi.

- Seluruh Keluarga Besar Laboratorium Informatika dan KomputerTeknik Elektro Universitas Brawijaya atas kerjasama dan pengalaman berharga selama masa jabatan sebagai asisten laboratorium,
- Teman-teman Konsentrasi Rekayasa Komputer angkatan 2013, Nardo, Huda, Muslichin, Erza, Ramadhan, Firman, Pretty, Dian,
- Seluruh pihak yang telah membantu yang tidak bisa penulis sebutkan satu persatu, terimakasih atas bantuan dan dukungannya.

Semoga Allah SWT mencatat segala bantuan dari semua pihak yang turut membantu dalam penyelesaian skripsi ini sebagai amalan ikhlas yang akan bermanfaat kelak. Penulis menyadari bahwa skripsi ini masih perlu banyak perbaikan. Penulis sangat mengharapkan kritik dan saran agar kedepannya skripsi ini dapat dikembangkan lebih lanjut. Penulis berharap skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan dan teknologi serta bagi masyarakat.

Malang, 11 Desember 2018

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

RINGKASAN

Nararya Andika Sujarwo, Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya, Desember 2018 *Metode Failover Untuk Mengatasi Power Loss Pada Node Wireless Sensor Network (WSN)*, Dosen Pembimbing : Adharul Muttaqin.,S.T.,M.T.

Wireless Sensor Network (WSN) atau jaringan sensor nirkabel yang terdiri dari beberapa sensor yang saling bekerjasama untuk memonitor fisik dan kondisi pada ruangan ataupun lingkungan. Perancang WSN harus mempertimbangkan kendala sumber daya termasuk jumlah energi yang terbatas, jangkauan komunikasi jarak pendek, *bandwidth* rendah, serta pemrosesan dan penyimpanan yang terbatas. *Node* WSN pada umumnya memiliki enam komponen yang terdiri dari *power supply*, sensor, *processor*, *storage*, *transceiver* dan *actuator*. Kegagalan pada *node* WSN dapat disebabkan oleh salah satu komponen tersebut. Salah satu contoh dari kegagalan pada *node* WSN adalah kegagalan pada *power supply* akibat dari tidak adanya pasokan daya ataupun *node* dalam perbikan, sehingga tidak terjadi pengiriman data. Metode *failover* dapat diterapkan untuk meningkatkan keandalan dan ketersediaan *node* WSN dengan memberikan sistem cadangan yang mengambil alih layanan pada sistem utama. Sistem tetap berjalan ketika sistem utama gagal atau mati karena digantikan oleh sistem cadangan.

Skripsi ini menerapkan metode *failover* pada *node client* dan *node access point.* Penerapan metode *failover* pada *node* WSN digunakan mekanisme prosesor *dual* redundansi yang menggunakan sistem utama dan sistem cadangan. Prosesor kedua memiliki catu daya yang hanya digunakan ketika catu daya utama dimatikan. Pada sistem digunakan Arduino Uno sebagai prosesor dan ESP8266-01 sebagai radio transceiver.

Pengujian *failove*r pada *client* dilakukan dengan menerapkan kondisi *failover* hanya pada *node client*, didapatkan rata-rata kecepatan sistem *failover* 102.2 ms dengan waktu minimum 19 ms dan maksimum 149 ms. Pada sistem ini pengiriman dilakukan setiap 100 ms sehingga terdapat kehilangan rata-rata 2 data. Pengujian *failover* pada *acces point* dilakukan dengan membuat kondisi *failover* pada *acces point*. *Client* dihubungkan pada komputer dan didapatkan waktu *failove*r selama 0 ms. Pengujian *failback* dilakukan dengan mengembalikan sistem setelah terjadinya *failover*. Pada pengujian *client* didapatkan waktu rata-rata *failback* 318.4 ms dengan waktu minimum pengembalian sistem adalah 277 ms dan waktu maksimum pengembalian sistem 357 ms. Pengujian *failback* pada *access point* didapatkan waktu *failback* dari sistem adalah 0 ms.

Kata Kunci: *Wireless Sensor Network* (WSN), *failover, high availability*

SUMMARY

Nararya Andika Sujarwo, Department of Electrical Engineering, Faculty of Engineering, Brawijaya University, December 2018 Failover Method to Overcome Power Loss in Wireless Sensor Network (WSN) Nodes, Advisor: Adharul Muttaqin., S.T., M.T.

Wireless Sensor Network (WSN) consists of several sensors that connected to monitor the physical and conditions in the room or environment. The WSN designer must consider resource constraints including the quantity of energy, nearby communication range, low bandwidth, and limited processing and storage. The WSN node generally has six components consisting of power supply, sensor, processor, storage, transceiver and actuator. Failure on the WSN node can be caused by one of these components. One example of a failure in the WSN node is a failure in the power supply due to the absence of power supplies or nodes in the maintenance, so data transmission does not occur. A failover method can be applied to improve the reliability and availability of WSN nodes by providing a backup system that takes over services on the main system. The system keeps running when the main system fails or shut down because it is replaced by a backup system.

This final project applies the failover method on the client node and the access point node. Then failover method on the WSN node is implemented using dual processor mechanism that uses the main system and backup system. The power supply of second processor only used when the main power supply is turned off. The system use Arduino Uno as processor and ESP8266-01 as radio transceiver.

Failover test on the client is done by applying failover conditions only on client nodes, obtained an average failover system speed of 102.2 ms with a minimum of 19 ms and a maximum of 149 ms. In this system the transmission is done every 100 ms so there is an average loss of 2 data. Failover testing on the access points is done by making failover conditions on the access points. The client connected to the computer and obtained 0 ms failover time. Failback testing is done by returning the system after a failover. On the client test, the average failback time is 318.4 ms with a minimum return time of 277 ms and a maximum system return time of 357ms. The failback test on the access point shows that the failback time from the system is 0 ms.

Keywords: Wireless Sensor Network (WSN), failover, high availability

BAB I
PENDAHULUAN

1.1 Latar Belakang

Besaran yang menyatakan derajat panas dingin suatu benda adalah suhu. Suhu merupakan satu hal yang dapat menjelaskan mengenai kondisi lingkungan ataupun ruangan. Kebutuhan akan data mengenai kondisi ruangan sekitar telah mendorong manusia untuk membuat alat yang bisa mengetahui kondisi ruangannya. Suhu suatu ruangan pada suatu titik tertentu dapat berbeda-beda dan dapat terpengaruh oleh benda yang menghasilkan panas. Oleh karena itu dibutuhkan suatu perangkat yang dapat mengukur suhu didalam ruangan pada titik tertentu dan diluar ruangan. Dari hal tersebut akan dibuat sebuah perangkat yang dapat mengukur suhu dan kelembapan sekaligus dengan menggunakan mikrokontroler sebagai pusat kendalinya. Untuk memudahkan dalam melakukan *monitoring* suhu dan kelembapan ruangan, maka diperlukannya suatu alat yang dapat menginformasikan keadaan tersebut secara terus menerus (*real time*) yaitu *Wireless sensor network.*

Wireless Sensor Network (WSN) merupakan jaringan nirkabel yang terdiri dari beberapa sensor yang saling bekerja sama untuk memonitor fisik dan kondisi pada ruangan ataupun lingkungan seperti temperatur, air, suara, polusi udara, dan gas dititik yang berbeda. *Wireless sensor network* (WSN) secara umum terdiri dari *base station* atau ("*gateway*") yang dapat berkomunikasi melalui jaringan dengan sejumlah sensor nirkabel. Simpul sensor nirkabel mengumpulkan data, dan mentransmisikannya ke *gateway* secara langsung atau bila perlu, menggunakan *node* sensor nirkabel lainnya untuk meneruskan data ke*gateway*. Data yang dikirimkan kemudian dipresentasikan ke sistem melalui koneksi *gateway*. *Wireless Sensor Network* (WSN) terdiri dari perangkat independen yang tersebar secara *spasial* yang menggunakan sensor untuk memantau kondisi fisik atau lingkungan. Perangkat independen ini, yang dikenal sebagai *router* dan *end node*, bersamaan dengan *gateway* bergabung untuk menciptakan sistem WSN. Dari berbagai macam konfigurasi dan metode telah tersedia saat ini, dapat membangun perangkat *Wireless Sensor Network* (WSN) dengan biaya murah karena tidak membutuhkan perangkat khusus. Seorang perancang *Wireless Sensor Network* (WSN) harus berurusan dengan kendala sumber daya termasuk jumlah energi yang terbatas, jangkauan komunikasi pendek, bandwidth rendah, dan pemrosesan dan penyimpanan yang terbatas

disetiap *node*. Kendala desain tergantung pada aplikasi dan berdasarkan lingkungan yang dipantau. Lokasi dimana WSN diimplementasikan memainkan peran penting untuk mempertimbangkan menentukan ukuran jaringan, strategi penyebaran, dan arsitektur jaringan. Ukuran jaringan bervariasi dari beberapa *node* untuk lingkungan, dalam ruangan hingga ribuan *node* untuk mencakup area yang lebih luas.

Metode *failover* atau *high availability* merupakan suatu alternatif jika memiliki lebih dari satu koneksi untuk menjaga ketersediaan koneksi. Metode *failover* ini dapat secara otomatis bekerja pada *line* yang mengalami putus koneksi. Definisi *failover* dalam istilah komputer adalah kemampuan sebuah sistem untuk dapat ber*pin*dah secara manual maupun otomatis jika salah satu sistem mengalami kegagalan sehingga menjadi *backup* untuk sistem yang mengalami kegagalan. Metode *failover* menyediakan solusi *high availability* dimana jika terjadi kegagalan pada sistem *hardware* seperti *power supply* mati yang menyebabkan sistem mati, dengan secara otomatis fungsi dari sistem cadangan akan mengambil alih fungsi dari sistem yang mati, sehingga pada saat user mengakses tidak mengetahui jika terjadi kegagalan akses pada sistem, karena proses yang dilakukan pada sistem utama yang gagal atau mati akan dilanjutkan oleh sistem *backup*. Untuk menjadi *high available* dalam layanan maka fungsi *failover* menjadi peran utama dalam menjaga kestabilan dan kontinuitas saat salah satu sistem mengalami masalah.

Skenario *failover* harus diterapkan untuk meningkatkan keandalan dan ketersediaan *node Wireless Sensor Network* (WSN). Salah satu contoh penyebab kegagalan layanan pada *node Wireless Sensor Network* (WSN) adalah sistem kehilangan daya, baik karena pemeliharaan atau karena kehilangan pasokan listrik. *Failover* biasanya dioperasikan secara otomatis tanpa peringatan. Dengan menerapkan metode *failover*, *node* akan tetap dapat mengirim data sensor dan dapat memberikan kesempatan untuk melakukan perawatan pada sistem utama.

Penerapan mekanisme prosesor dua *redundansi* pada *node Wireless Sensor Network* (WSN) dalam skenario *failover* pada kegagalan daya atau *power loss* dianalisis. Sistem utama menggunakan prosesor pertama dan sistem cadangan menggunakan prosesor kedua. Prosesor kedua memiliki catu daya yang hanya digunakan ketika catu daya utama dimatikan. Pada sistem digunakan Arduino Uno sebagai prosesor, dan ESP8266-01 sebagai radio *transceiver*.

1.2 Rumusan Masalah

Dengan identifikasi masalah yang telah di jelaskan diatas maka penulis membuat sistem *Wireless Sensor Network* (WSN) dengan menggunakan arduino dan menerapkan metode *failover*. Rumusan yang di rumuskan pada tugas akhir ini adalah :

1. Bagaiman mengatasi kegagalan pemrosesan data pada *node Wireless Sensor Network* (WSN) akibat kehilangan daya listrik ?
2. Bagaimana menerapkankan metode *failover* pada *node Wireless Sensor Network* (WSN) yang menggunakan arduino dan modul ESP8266 ?

1.3 Batasan Masalah

- Prosesor yang digunakan adalah mikrokontroler arduino UNO.
- Pengirim dan penerima data pada sistem *Wireless Sensor Network* (WSN) menggunakan modul ESP8266-01.
- Sistem dirancang untuk mengukur suhu menggunakan sensor DHT11.
- Pengujian dilakukan pada *node* arduino yang telah dirancang.
- Setiap *node* diuji satu persatu dengan memberikan prosesor cadangan.
- Pada sistem terdapat satu server dan satu *client*.
- Sistem yang dirancang, untuk mengatasi kegagalan pada *node Wireless Sensor Network* (WSN) akibat kehilangan daya listrik, disimulasikan dengan mematikan prosesor utama.
- Pengujian yang dilakukan untuk mengetahui berapa lama waktu terjadinya *failover* dan pengembalian sistem ketika arduino utama dihidupkan kembali.
- Pengujian yang dilakukan pada sistem *Wireless Sensor Network* (WSN) ini tidak melihat seberapa lama waktu *up-time* dari sistem.

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah menganalisis sistem yang mnerapkan metode *failover* pada *node Wireless Sensor Network* (WSN) yang menggunakan arduino untuk mengatasi *power loss*. Pada penelitian ini metode *failover* akan diterapkan pada mikrokontroler arduino, dimana membuat dua buah *node Wireless Sensor Network* (WSN) yang disetiap *node*nya terdapat dua arduino dan satu modul *wireless*, yang mana salah satu arduino adalah cadangan jika terjadi masalah dan akan melakukan *takeover* apabila terjadi maslah pada arduino pertama.

1.5 Manfaat Penelitian

Penelitian ini di harapkan dapat bermanfaat untuk berbagai pihak. Manfaat dari penelitian ini sebagai berikut :

- Bagi penulis
  - Menjadikan pijakan awal bagi penulis untuk membuat penelitian yang berhubungan dengan arduino dan *Wireless Sensor Network* (WSN).
  - Untuk mengaplikasikan ilmu yang telah diperoleh selama menempuh pendidikan di perguruan tinggi dengan membuat laporan penilitian.
- Bagi pembaca
  - Menambah wawasan tentang *Wireless Sensor Network* (WSN).
  - Menjadi bahan refrensi untuk penelitian yang berkaitan dengan *Wireless Sensor Network* (WSN) dan arduino.
  - Dapat melakukan pengembangan *Wireless Sensor Network* (WSN)

1.6 Sistematika Pembahasan

Sistematika pembahasan dari penyusunan penelitian ini digunakan sebagai berikut:

BAB I Pendahuluan

Bab Pendahuluan membahas latar belakang penulisan tugas akhir, rumusan permasalahan, tujuan tugas akhir, ruang lingkup dan batasan permasalahan yang di tangani, manfaat penelitian serta sistematika pembahasan.

BAB IITinjauan Pustaka

Bab tinjauan pustaka berisi teori-teori yang menguraikan konsep-konsep yang mendasari dan mendukung masalah-masalah tugas akhir.

BAB III Metodologi Penelitian

Bab metodologi penelitian berisi tentang metode-metode yang dipakai untuk melakukan perancangan, pengujian dan analisa data.

BAB IV Pengujian Dan Analisis

Bab pengujian dan analisis memuat aspek pengujian meliputi penjelasan tentang cara pengujian dan hasil pengujian. Aspek analisis meliputi penilaian atau komentar terhadap hasil-hasil pengujian. Pengujian dan analisis ini terhadap alat yang telah di realisasikan berdasarkan masing-masing blok dan sistem secara keseluruhan.

BAB V Penutup

Bab penutup terdapat kesimpulan dan saran memuat intisari hasil pengujian dan menjawab rumusan masalah serta memberikan rekomendasi untuk perbaikan kualitas penelitian di masa yang akan datang.

BAB II
TINJAUAN PUSTAKA

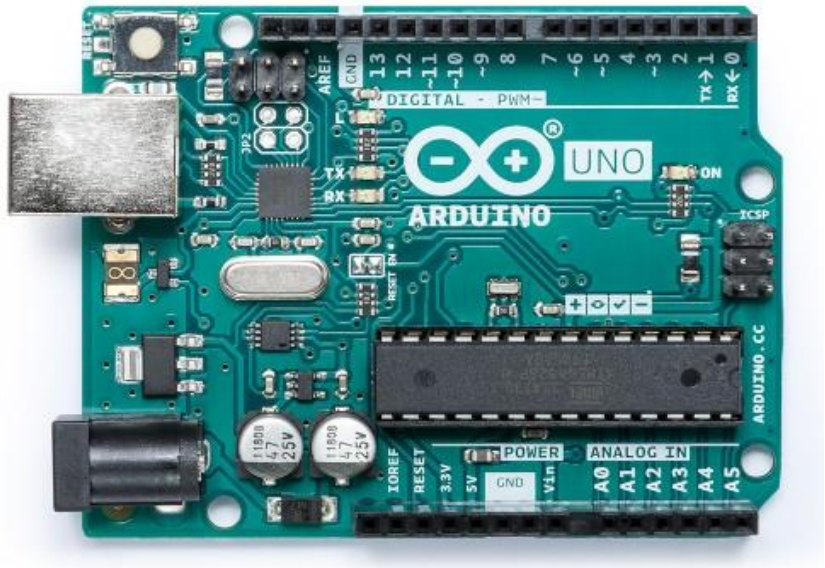
Pada bab ini diuraikan pustaka-pustaka yang berkaitan dengan penelitian, yang meliputi pembahasan tentang Arduino, komunikasi serial, metode *failover*, *availability* sistem, *wireless sensor network*, ESP8266-01, sensor DHT11 dan *Relay*.

2.1 Arduino

Dalam buku "*Getting Started with Arduino"*, arduino dituliskan sebagai sebuah *platform* komputasi fisik yang *open source* pada *board input output* sederhana, yang dimaksud komputasi fisik disini adalah sebuah sistem fisik yang interaktif dengan penggunaan *sotware* dan *hardware* yang dapat mendeteksi dan merespon situasi dan kondisi yang ada pada dunia nyata. Sedangkan dari situs resminya diarduino.cc, arduino didefenisikan sebagai sebuah *platform* elektronik yang *open source*, berbasis pada *software* dan *hardware* yang fleksibel dan mudah digunakan, yang ditujukan untuk para seniman, desainer, hobi dan setiap orang yang tertarik dalam membuat objek atau lingkungan yang interaktif. Nama arduino disini tidak hanya dipakai untuk menamai board rangkaiannya saja, tetapi juga untuk menamai bahasa dan *software* pemrogramannya, serta lingkungan pemrogramannya atau IDE-nya (IDE = *intergrated development environment*). (Artanto, 2012)

Arduino Uno adalah *board* mikrokontroler menggunakan chip ATmega328P. Arduino uno memiliki 14 *pin input* atau *output* digital (yang 6 dapat digunakan sebagai *output* PWM), 6 *input analog*, Kristal *Quartz* 16 MHz, koneksi USB, *power jack*, *header* ICSP dan tombol *reset*. Arduino berisi semua yang dibutuhkan untuk mendukung mikrokontroler, untuk memulai arduino dengan menghubungkan arduino ke komputer dengan kabel USB atau menyalakan arduino dengan adaptor AC-ke-DC atau baterai. (Arduino, 2018)

Arduino dilengakpi dengan *static random access memory* (SRAM) berukuran 2KB untuk memegang data *flash memory* berukuran 32KB, dan *erasable programmable read-only memory* (EEPROM). SRAM digunakan untuk menampung data atau hasil pemrosesan data selama arduino menerima pasokan catu daya. *Falsh memory* untuk menaruh program yang di buat. EEPROM digunakan untuk menaruh program bawaan dari arduino uno dan sebagian lagi dapat dimanfaatkan sebagai penyimpan data secara permanen. (Kadir, 2015)



Gambar 2.1 Arduino UNO
Sumber: arduino.cc

2.1.1 *Power*

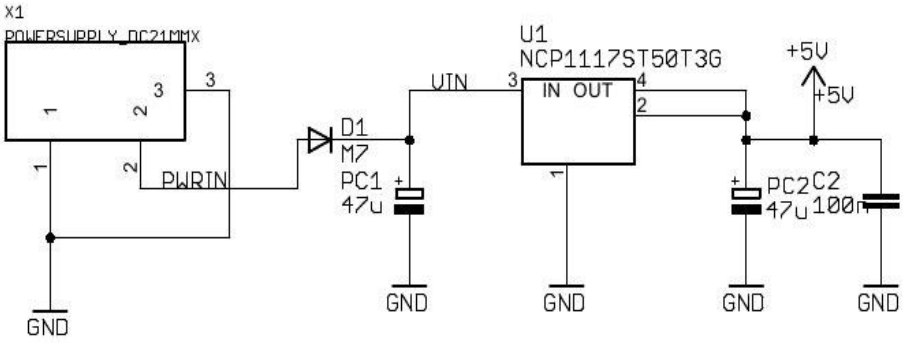
Mengaktifkan Arduino Uno dapat dilakukan melalui koneksi USB atau dengan catu daya eksternal lainnya. Sumber daya dipilih secara otomatis oleh Arduino. Sumber daya eksternal dapat berasal baik dari USB, adaptor AC-DC atau baterai. Adaptor dapat dihubungkan dengan menghubugkan *power jack* 2,1 mm yang bagian tengahnya terminal positif ke *jack* sumber tegangan pada *board*. Jika tegangan berasal dari baterai dapat langsung dihubungkan melalui *header pin* Gnd dan *pin* Vin dari konektor *POWER*.

Board Arduino Uno dapat beroperasi dengan pasokan daya dari *power jack* 6 Volt sampai 20 Volt. Jika diberi tegangan kurang dari 7 Volt, maka *pin* 5 Volt mungkin menghasilkan tegangan kurang dari 5 Volt dan ini membuat *board* menjadi tidak stabil. Jika sumber tegangan menggunakan lebih dari 12 Volt, regulator tegangan mengalami panas berlebihan dan bisa merusak *board*. Rentang sumber tegangan yang dianjurkan adalah 7 Volt sampai 12 Volt.

- Vin. Adalah *input* tegangan untuk *board* Arduino ketika menggunakan sumber daya eksternal (sebagai pembanding tegangan 5 Volt dari koneksi USB atau sumber daya terregulator lainnya). Anda dapat memberikan tegangan melalui *pin* ini, atau jika memasok tegangan untuk *board* melalui *jack power*, kita bisa mengakses/mengambil tegangan melalui *pin* ini.

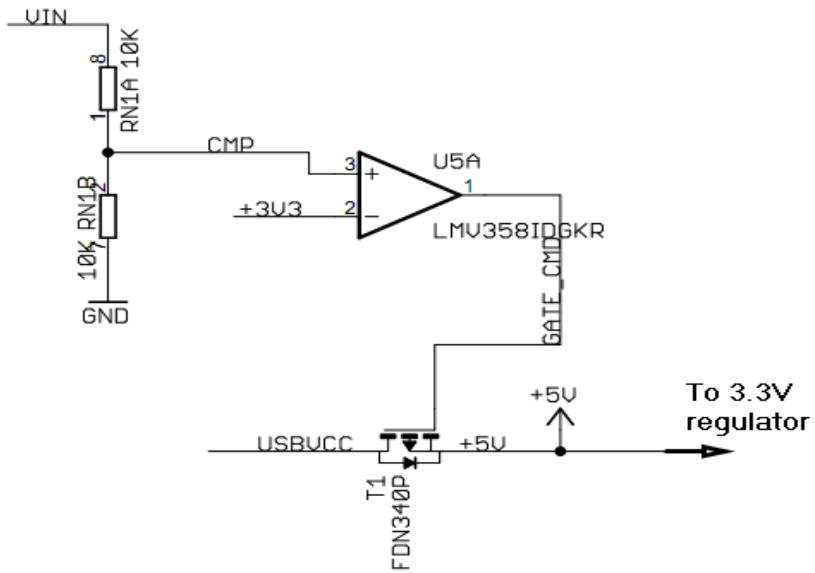
- 5V. Sebuah *pin* yang mengeluarkan tegangan ter-regulator 5 Volt, dari *pin* ini tegangan sudah diatur (ter-regulator) dari regulator yang tersedia (*built-in*) pada *board*. Arduino dapat diaktifkan dengan sumber daya baik berasal dari *jack power* DC (7-12 Volt), konektor USB (5 Volt), atau *pin* VIN pada board (7-12 Volt). Memberikan tegangan melalui *pin* 5V atau 3.3V secara langsung tanpa melewati regulator dapat merusak *board* Arduino.
- 3V3 : Sebuah *pin* yang menghasilkan tegangan 3,3 Volt. Tegangan ini dihasilkan oleh regulator yang terdapat pada *board* arduino. Arus maksimum yang dihasilkan adalah 50 mA.
- GND. *Ground Pin*.
- IOREF. *Pin* ini pada *board* Arduino berfungsi untuk memberikan referensi tegangan yang beroperasi pada mikrokontroler. Sebuah perisai (*shield*) dikonfigurasi dengan benar untuk dapat membaca *pin* tegangan IOREF dan memilih sumber daya yang tepat atau mengaktifkan penerjemah tegangan (*voltage translator*) pada *output* untuk bekerja pada tegangan 5 Volt atau 3,3 Volt.
- USB. *Port* USB pada Arduino memiliki dua fungsi, yaitu sebagai *port* untuk komunikasi *serial* dan sebagai catu daya untuk menyalakan *board* Arduino. Saat Arduino terkoneksi dengan komputer melalui *port* USB, Arduino secara otomatis mendapatkan *supply* sebesar 5V dari komputer. Hal ini akan memudahkan ketika melakukan pemrograman dan pengujian proyek yang sedang buat, karena tidak perlu lagi menambahkan *power supply* eksternal untuk proyek. Namun perlu diperhatikan bahwa kapasitas *power supply* dari USB sangatlah terbatas, hanya 500 mA saja, sehingga jika proyek yang dibuat memerlukan daya yang lebih besar – misalnya untuk menggerakkan motor – maka tetap membutuhkan *power supply* eksternal.
- Jack 2.1mm. *Jack* 2.1mm pada Arduino digunakan sebagai masukan dari *power supply* eksternal. Arduino dapat menerima tegangan *power supply* sebesar 7 hingga 20 VDC, dengan tegangan yang direkomendasikan adalah 12 VDC. Polaritas *power supply* adalah positif pada *pin* tengahnya, namun jika kita secara tidak sengaja memasang *jack power supply* dengan polaritas terbalik, Arduino akan tetap aman, karena *board* Arduino telah dilengkapi dengan dioda sebagai pengaman terhadap kesalahan polaritas *power supply*. (Arduino, 2018)

Proses jika menggunakan dua *power supply* pada Arduino dengan Menggabungkan dua *power supp*ly menggunakan *power supply* dari *jack* sekaligus menghubungkan kabel USB ke komputer.



Gambar 2.2 *Supply* Dari *Jack* 2.1mm
Sumber: arduino.cc

Gambar 2.2 memperlihtakan rangkaian *power supply* dari jack 2.1mm. *Supply* daya melewati dioda D1 (sebagai pengaman jika polaritas *power supply* yang digunakan ternyata salah atau terbalik), dan masuk ke regulator NCP1117 yang akan menurunkan tegangan ke 5V untuk men*supply* komponen-komponen yang memerlukan tegangan 5V. Rangkaian untuk *power supply* dari USB ditunjukkan pada gambar 2.3 sebagai berikut.



Gambar 2.3 *Supply* Dari USB *port*
Sumber: arduino.cc

Power supply dari USB masuk ke *P-mosfet* T1 yang berfungsi sebagai saklar, dan kemudian masuk ke jalur +5V (menjadi satu dengan jalur keluaran 5V seperti ditunjukkan pada rangkaian sebelumnya).

Op-amp LMV358 yang bekerja sebagai komparator memberikan *input* pada *mosfet* agar bekerja sebagai saklar. Tegangan *input* op-amp terhubung dengan jalur 3.3V dan *input* positif diambil dari rangkaian pembagi tegangan, yang besarnya setengah dari tegangan pada jalur VIN. Saat tegangan VIN > 6.6V maka *input* positif op-amp menjadi lebih besar dari *input* negatif, dan op-amp mematikan mosfet. Saat tegangan VIN < 6.6V maka *input* positif op-amp menjadi lebih rendah daripada *input* negatif, dan op-amp mengaktifkan mosfet. Jalur VIN berada setelah dioda *input* sehingga tegangannya 0.6V lebih rendah daripada tegangan *power supply* eksternal. Jika tegangan *power supply* eksternal lebih besar dari 6.6V+0.6V (dimana 0.6V adalah drop tegangan pada dioda D1), maka power supply dari USB diputus (karena mosfet tidak aktif) dan tegangan diambil dari *power supply* eksternal yang lain. Dengan demikian, tidak ada perbedaan jika kita menghubungkan dan mencabut kabel USB pada saat Arduino terhubung dengan *power supply* eksternal yang lain. *Power supply* ber*pin*dah ke USB hanya jika tegangan *power supply* eksternal yang lain turun dibawah level tegangan yang diperbolehkan (sekitar 6.6V+0.6V=7.2V). (Abrams, 2014)

2.1.2 Memori

Prosessor ATmega328 memiliki memori sebesar 32 KB yang mana sebesar 0,5 KB digunakan untuk menyimpan file *bootloader*. ATmega328 juga memiliki 2 KB SRAM dan 1 KB EEPROM (yang dapat dibaca dan ditulis dengan perpustakaan EEPROM).

- Memori *Flash*: memori yang digunakan untuk menyimpan program. Memori *flash* juga dikenal sebagai program ROM, algoritma-algoritma yang dibuat dalam program disimpan didalam memori ROM. Memori ROM tidak hilang ketika arduino kehilangan daya.
- Memori SRAM: memori yang digunakan untuk menyimpan data, berbeda dengan ROM yang menyimpan program memori bertipe RAM ini digunakan untuk menyimpan data. Data dalam memori ini akan hilang ketika arduino kehilangan daya.
- Memori EEPROM: memori yang digunakan untuk menyimpan data yang sifatnya tidak sering dihapus, mikrokontroler atmrl di lengkapi sejumlah kecil ROM dan dapat di akses menggunakan pustaka EEPROM pada arduino. (Arduino, 2018)

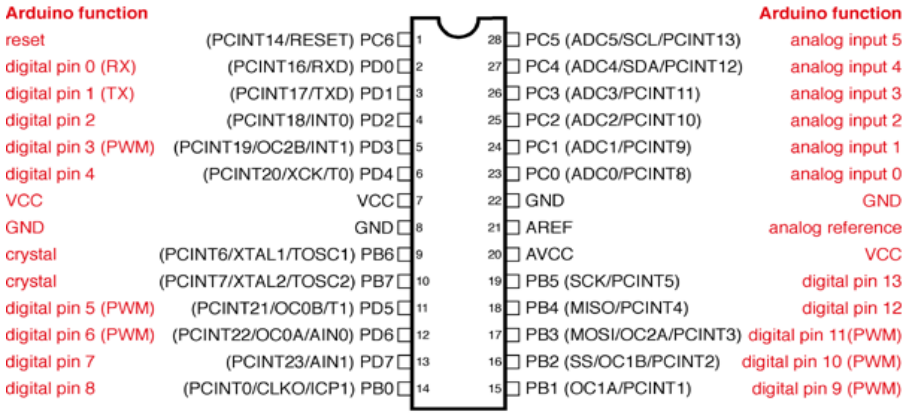
2.1.3 *Input dan Output*

Masing-masing dari 14 *pin* digital pada Arduino Uno dapat digunakan sebagai *input* atau *output* dengan menggunakan fungsi *pinMode()* , *digitalWrite()* , dan *digitalRead().* Semua *pin* beroperasi pada tegangan 5 volt. Setiap *pin* dapat memberikan atau menerima arus maksimum 40 mA dan memiliki resistor *pull-up internal* (terputus secara default) sebesar 20-50 kOhm. Selain itu beberapa *pin* memiliki fungsi khusus, yaitu:

- *Serial* : 0 (RX) dan 1 (TX). Digunakan untuk menerima (RX) dan mengirimkan (TX) TTL data *serial*. *Pin* ini terhubung ke *pin* korespondensi dari *chip* ATmega8U2 *Serial* USB-to-TTL.
- *External Interrupt* (*Interupsi Eksternal*): *Pin* 2 dan *pin* 3 ini dapat dikonfigurasi untuk memicu sebuah interupsi pada nilai yang rendah, meningkat atau menurun, atau perubahan nilai.
- PWM : *Pin* 3, 5, 6, 9, 10, dan 11. Menyediakan *output* PWM 8-bit dengan fungsi *analogWrite()*.
- SPI : *Pin* 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). *Pin* ini mendukung komunikasi SPI menggunakan perpustakaan SPI .
- LED : *Pin* 13. Tersedia secara *built-in* pada *board* Arduino Uno. LED terhubung ke *pin* digital 13. Ketika *pin* diset bernilai *HIGH*, maka LED menyala, dan ketika *pin* diset bernilai *LOW*, maka LED padam.

Arduino Uno memiliki 6 *pin* sebagai *input analog*, diberi label A0 sampai dengan A5, yang masing-masing menyediakan resolusi 10 bit (yaitu 1024 nilai yang berbeda). Secara default *pin* ini dapat diukur/diatur dari mulai Ground sampai dengan 5 Volt, juga memungkinkan untuk mengubah titik jangkauan tertinggi atau terendah mereka menggunakan *pin* AREF dan fungsi *analog Reference ().* Selain itu juga, beberapa *pin* memiliki fungsi yang dikhususkan, yaitu:

- TWI : *Pin* A4 atau SDA dan *pin* A5 atau SCL yang mendukung komunikasi TWI menggunakan perpustakaan Wire.
- AREF : Referensi tegangan untuk *input analog*. Digunakan dengan fungsi *analogReference()*.
- RESET : Jalur *LOW* ini digunakan untuk me-reset (menghidupkan ulang) mikrokontroler. Jalur ini biasanya digunakan untuk menambahkan tombol reset pada *shield* yang menghalangi *board* utama Arduino.



Gambar 2.4 Atmega168 *Pin Mapping*
Sumber: arduino.cc

2.1.4 *Communication*

Arduino Uno memiliki sejumlah fasilitas untuk berkomunikasi dengan komputer, dengan Arduino lain, atau dengan mikrokontroler lainnya. ATmega328 menyediakan komunikasi *serial* UART TTL (5 Volt), yang tersedia pada *pin* digital 0 (RX) dan *pin* 1 (TX). Sebuah chip ATmega16U2 yang terdapat pada *board* digunakan sebagai media komunikasi *serial* melalui USB dan muncul sebagai *COM Port Virtual* (pada Device komputer) untuk berkomunikasi dengan perangkat lunak pada komputer. *Firmware* 16U2 menggunakan *driver* standar USB *COM*, dan tidak membutuhkan *driver* eksternal. Namun pada sistem operasi Windows, file .inf masih dibutuhkan. Perangkat lunak Arduino termasuk didalamnya *serial* monitor memungkinkan data tekstual sederhana dikirim dari *board* Arduino. LED RX dan TX yang tersedia pada *board* berkedip ketika data sedang dikirim atau diterima melalui chip USB-to-*serial* yang terhubung melalui USB komputer (tetapi tidak untuk komunikasi *serial* seperti pada *pin* 0 dan 1).

Sebuah *library SoftwareSerial* memungkinkan komunikasi *serial* pada beberapa *pin* digital Uno. ATmega328 juga mendukung komunikasi I2C (TWI) dan SPI. Perangkat lunak Arduino termasuk perpustakaan *Wire* digunakan untuk menyederhanakan penggunaan bus I2C. Komunikasi SPI, menggunakan *library* SPI. (Arduino, 2018)

2.1.5 Pemrograman

Arduino Uno dapat diprogram dengan software Arduino (Arduino IDE). Arduino Uno sudah tersedia dengan *bootloader* yang memungkinkan pemrogram untuk meng-upload kode baru tanpa menggunakan program *hardware* eksternal. Hal ini karena komunikasi yang terjadi

menggunakan protokol asli STK500. Pemrogram juga dapat melewati (*bypass*) *bootloader* dan program mikrokontroler melalui *pin header* ICSP (*In-Circuit Serial* Programming).

Chip ATmega16U2 (atau 8U2 pada *board* Rev. 1 dan Rev. 2) *source code firmware* tersedia. ATmega16U2/8U2 dapat dimuat dengan *bootloader* DFU, yang dapat diaktifkan melalui:

- Pada *board* Revisi 1: Menghubungkan *jumper* solder dibagian belakang *board* (dekat dengan *map* Italia) dan kemudian akan me-reset 8U2.
- Pada *board* Revisi 2: Ada resistor yang menghubungkan jalur HWB 8U2/16U2 ke *ground*, sehingga lebih mudah untuk dimasukkan ke dalam mode DFU.

Kemudian Anda dapat menggunakan Atmel FLIP *software* (sistem operasi *Windows*) atau DFU program (sistem operasi Mac OS X dan Linux) untuk memuat *firmware* baru. Atau pemrogram dapat menggunakan *pin header* ISP dengan program eksternal (*overwrite* DFU *bootloader*). (Arduino, 2018)

2.1.6 Mode *Sleep* Arduino

Mode *sleep* adalah mode khusus dari prosesor yang menghentikan eksekusi program normal dan mematikan komponen internal untuk mengurangi konsumsi daya. Setelah prosesor tertidur, prosesor hanya dapat dibangunkan oleh peristiwa tertentu seperti *interupsi* eksternal dari penekanan tombol, atau penghitung waktu yang menghitung periode waktu.

Pada bagian depan *hardware*, Arduino dilengkapi dengan dua port *interupsi*: *pin* digital 2 dan 3. Arduino dapat menggunakan *pin-pin interupsi* untuk *interupsi* bangun dan melanjutkan eksekusi kode. Bahkan dimungkinkan untuk mengeksekusi kode khusus tergantung pada *pin* mana yang memicu (*interupsi*). USART (port *serial*) juga membangunkan Arduino, agar ini berfungsi, Arduino harus dalam POWER_MODE_IDLE, satu-satunya mode *sleep* yang tidak menonaktifkan USART. Meskipun mode ini tidak memberikan penghematan daya yang besar, dapat digunakan fungsi yang disediakan *avr / power.h* (*power_adc_disable(), power_spi_disable(), power_timer0_disable(), power_timer1_disable(), power_timer2_disable(), power_twi_disable ()*) untuk menonaktifkan modul perangkat keras untuk mencapai penghematan daya yang lebih besar.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Gambar 2.5 Mode *sleep* Arduino
Sumber: learn.adafruit.com

Gambar 2.5 diatas adalah tabel mode *sleep* dari *datasheet* prosesor ATmega328P (prosesor yang menggerakkan Arduino Uno R3 dan Nano v3). Perhatikan bahwa mode *sleep* yang berbeda menonaktifkan berbagai *clock* dan komponen *internal* untuk mengurangi konsumsi daya, yang juga menarik adalah apa yang dapat membangunkan prosesor dari setiap mode tidur, seperti *interupsi* atau *timer*.

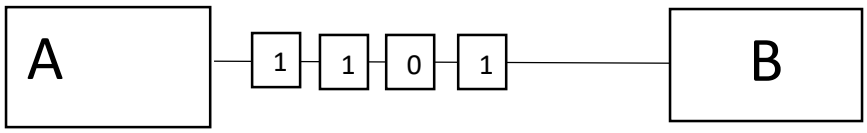
Ketika arduino dalam *SLEEP*_MODE_PWR_DOWN satu-satunya cara untuk membangunnya adalah dengan *interupsi* tingkat pada *pin* 2 atau 3, atau *Pin Change interrupt*. Level *interrupt* berarti *pin* harus ditahan dalam keadaan itu untuk jangka waktu tertentu sebelum *interupsi* dipicu. (Dicola, 2018)

2.2 Komunikasi *Serial*

Komunikasi *serial* adalah komunikasi yang pengiriman data secara berurutan dan bergantian. komunikasi ini mempunyai suatu kelebihan yaitu hanya membutuhkan satu jalur kabel, lebih sedikit apabila dibandingkan dengan komunikasi *paralel*. Pada prinsipnya komunikasi *serial* merupakan dimana pengiriman data dilakukan per-bit sehingga lebih lambat dibandingkan dengan komunikasi *paralel*, atau dengan kata lain komunikasi *serial* merupakan salah satu metode komunikasi data dimana hanya satu bit data yang dikirimkan melalui seuntai kabel pada suatu waktu tertentu.

Menurut Teguh Wahyuno dalam bukunya *Komunikasi Data*, pengiriman *serial* menimbulkan tiga masalah pokok untuk keperluan penyesuaian transmisi yaitu penyesuaian perbit, penyesuaian blok dan penyesuain waktu. Misalanya jikan akan dikirim data 1101, maka

agar data tersebut dapat dikirim dan diterima dengan benar, selang waktu yang digunakan oleh pengirim dan penerima harus sama. Jika penerima telah menerima penyesuaian bit, maka seharusnya juga harus segera menerima penyesuaian karakter. Selain itu penerima harus juga mengetahui awal dan akhir blok data. (Wahyuno, 2003)



Gambar 2.6 Komunikasi *Serial*
Sumber: Perancangan

Komunikasi *serial* ada dua macam, *asynchronus serial* dan *synchronus serial*. *Synchronus serial* adalah komunikasi dimana hanya ada satu pihak (pengirim atau penerima) yang menghasilkan *clock* dan mengirimkan *clock* itu secara bersama-sama dengan data. Contoh penggunaan *synchronus serial* terdapat pada transmisi data *keyboard*. *Asynchronus serial* adalah komunikasi dimana kedua pihak (pengirim atau penerima) masing—masing menghasilkan *clock* namun hanya data yang ditransmisikan, tanpa *clock*. Agar data yang di kirim sama dengan data yang diterima, maka kedua frekuensi *clock* harus sama dan harus terdapat *sinkornisasi*. Setelah adanya *sinkronisasi*, pengirim akan mengirimkan datanya sesuai dengan frekuensi *clock* pengirim dan penerima akan membaca data sesuai dengn frekuensi *clock* penerima. Contoh penggunaan *asynchronus serial* adalah pada *Universal Asynchronus Reciver Transmiter* (UART). (Wahyuno, 2003)

2.2.1 *Synchronous*

Synchronous transmission merupakan bentuk transmisi *serial* yang mentransmisikan data atau informasi secara terus menerus. Transmisi jenis ini sering menghadapi permasalahan, yaitu masalah *sinkornsasi* bit dan *sinkronsasi* karakter. Permasalahan utama dalam *sinkornsasi* adalah masalah waktu kapan *transmitter* mulai meletakkan bit-bit yang dikirim ke media transmisi dari kapan media harus mengetahui dengan tepat untuk mengambil bit-bit yang dikirim tersebut. Masalah ini dapat diatasi dengan *clock* yang ada di*transmitter* dan *clock* yang ada di*reciver*. *Clock* pada *transmitter* akan memberitahu kapan harus meletakkan bit-bit yang akan dikirim, misalnya jika diinginkan untuk mengirim dengan kapasitas 100 bps, *clock* di*transmitter* diatur untuk bekerja dengan kecepatan 100 bps dan *clock* di*reciver* juga harus diatur untuk mengambil dari jalur transmisi 100 kali tiap detiknya.

Permasalahan kedua dalam *synchronous transmission* adalah karakter *synchronization*. Permasalahan ini berupa penentuan sejumlah bit–bit mana saja yang merupakan bit-bit pembentuk suatu karakter. Hal ini dapat di atasi dengan memberikan karakter SYN. Umumnya dua atau lebih kontrol transmisi SYN didepan blok data yang dikirimkan. Bila hanya dipergunakan sebuah karakter kontrol *transmisi* kemungkinan dapat terjadi *false synchronization*. Untuk mencegah *false synchronization*, dua buah karakter kontrol SYN dapat digunakan diawal dari blok yang pertama, kemudian mengidentifikasi delapan bit berikutnya, kalau berupa karakter kontrol SYN yang kedua, maka setelah itu dimulai menghitung setiap delapan bit dan merangkai menjadi sebuah karakter. (Stallings, 2007)

2.2.2 *Asynchronous*

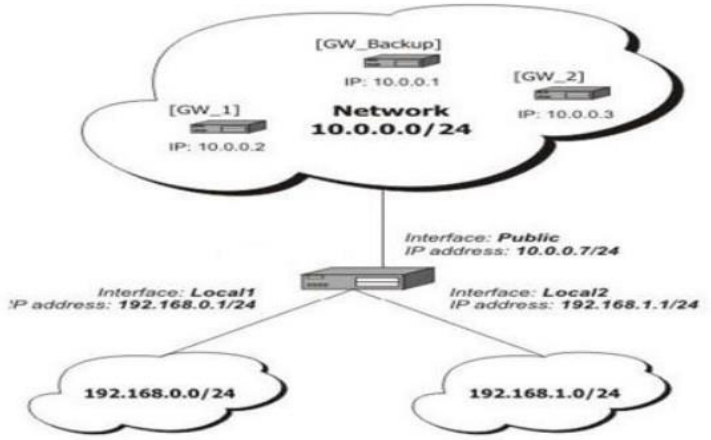
A*synchronous transmission* merupakan bentuk transmisi *serial* yang dalam mentransmisikan data atau informasi tidak secara terus menerus, dimana *transmitter* dapat mentransmisikan karakter-karakter pada *interval* waktu yang berbeda atau dengan kata lain tidak harus dalam waktu yang *sinkorn* antara pengiriman satu karakter dengan karakter berikutnya. Tiap-tiap karakter yang ditransmisikan sebagai satu kesatuan yang berdiri sendiri dan penerima harus dapat mengenal masing-masing karakter tersebut. Untuk mengatasi hal ini, maka masing-masing karakter di awali suatu bit tambahan, yaitu start bit yang berupa nilai bit 0 dan stop bit yang berupa nilai bit yang di letakkan pada akhir pada masing-masing karakter.

A*synchronous transmission* lebih aman dibandingkan dengan s*ynchronous transmission.* Pada a*synchronous transmission*, bila suatu kesalahan terjadi pada data yang di*transmisikan*, hanya merusak satu blok dari data. Akan tetapi, a*synchronous transmission* kurang efisien karena memerlukan bit-bit tambahan untuk tiap karakter yaitu start bit dan stop bit. (Stallings, 2007)

2.3 Metode *Failover*

Metode *failover* merupakan suatu alternatif jika memiliki lebih dari satu koneksi internet dan menjaga ketersediaan koneksi internet. Metode *failover* ini dapat secara otomatis bekerja pada *line* internet yang mengalami putus koneksi. Cara yang digunakan adalah penulis memasukkan *script* kedalam *iptables* dimana fungsi NAT yang berperan dalam mengoneksikan komputer ke internet, sehingga ketika koneksi *primary* putus maka *server* akan mengalihkan ke *line* yang masih hidup. Dari pengamatan yang penulis lakukan bahwa metode *failover* ini berjalan dengan baik dalam setiap percobaannya.

Definisi *failover* dalam istilah *computer internet working* adalah kemampuan sebuah sistem untuk dapat ber*pin*dah secara manual maupun otomatis jika salah satu sistem mengalami kegagalan sehingga menjadi *backup* untuk sistem yang mengalami kegagalan.



Gambar 2.7 Metode *failover*
Sumber: (Purnomo & Syafrizal, 2013)

Untuk mempermudah dan memperjelas maksud *failover* dapat melihat contoh gambar 2.7. Pada gambar gambar 2.7 dapat dilihat sebual *local area network* menggunakan lebih dari satu jalur jaringan isp. Jaringan lokal dengan ip 192.168.0.1/24 menggunakan *gateway* 1, sedangkan ip 192.168.1.1/24 menggunakan *gateway* 2. Ketika *gateway* 1 mengalami *disconnect* (putus) maka *gateway backup* akan menggantikan *gateway* 1. Jika *gateway* 1 sudah kembali normal maka jalur koneksi yang digunakan kembali menjadi *gateway* 1 dan begitu juga dengan *gateway* 2 apabila mengalami *disconnect* (putus).

Dengan begitu dapat disimpulkan bahwa tujuan dari metode *failover* adalah digunakan untuk menggantikan atau sistem *backup* koneksi *isp* yang terputus dengan koneksi *isp* yang lainnya. Metode *Failover* adalah proses peralihan ke sebuah komponen cadangan, elemen, atau operasi, sementara perbaikan untuk mengatasi gangguan sedang dijalankan. Prosedur *failover* menentukan kelangsungan operasional jaringan. Mekanisme *failover* dapat dirancang sehingga dapat sesegera mungkin bertindak setelah gangguan muncul. (Purnomo & Syafrizal, 2013)

2.4 *Availability* Sistem

Availability sistem dihitung dengan memodelkan sistem sebagai interkoneksi sistem secara *seri* dan *paralel*. Aturan berikut digunakan untuk memutuskan apakah komponen harus ditempatkan secara *seri* atau *paralel*. *Availability* sistem dianggap beroperasi secara *seri* jika

kegagalan salah satu bagian menghasilkan kegagalan kombinasi. Sistem gabungan hanya beroperasi jika kedua bagian tersedia. Dapat disimpulkan bahwa *availability* gabungan adalah produk dari *availability* dua bagian, yang mana *availability* gabungan dari dua komponen dalam *seri* selalu lebih rendah daripada ketersediaan komponen individualnya. *Availability* sistem dianggap beroperasi secara *paralel* jika kombinasi dianggap gagal ketika kedua bagian gagal, dapat disimpulkan bahwa *availability* gabungan adalah 1 - (kedua bagian tidak tersedia), *availability* gabungan dari dua komponen secara paralel selalu jauh lebih tinggi daripada *availability* komponen individualnya.

Menghitung *availability* komponen individu, nilai MTBF (*Mean time between failure atau uptime*) dan MTTR (*Mean time to repair atau downtime*). Dalam komponen perangkat keras, informasi MTBF dapat diperoleh dari perangkat keras yang mengirimkan data dan MTTR informasi didapatkan setelah sistem mengembalikan informasi, setelah MTBF dan MTTR diketahui, ketersediaan komponen dapat dihitung menggunakan rumus berikut (Rohani & Roosta, 2014):

$$A = \frac{MTBF}{MTBF + MTTR}$$

2.5 *Wireless Sensor Network* (WSN)

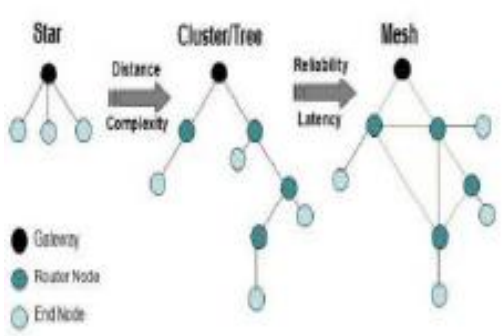
Wireless sensor network (WSN) sudah banyak diaplikasikan pada monitoring jarak jauh dan daerah yang tersebar sangat luas. Seperti pada jurnal "Programming *Wireless Sensor Network*: *Fundamental Concepts and State of the Art*"[2006], **Luca Mottola**, Univ of Trento, Italia. Tentang *fundamental* atau dasar dari teknologi WSN dan aplikasinya untuk pemantauan kondisi lingkungan dengan sensor yang tersebar cukup luas. WSN banyak diaplikasikan pada bidang monitoring kesehatan seseorang, otomasi industri, monitoring lingkungan dan konstruksi jembatan. Kemudian pengembangan kedepan dari WSN adalah *transfer* data getaran dari *Piezoelectric Sensor* dengan jarak yang cukup jauh dari *slave* node ke *master* node.

Wireless sensor network (WSN) secara umum terdiri dari *base station* (atau "*gateway*") yang dapat berkomunikasi melalui jaringan dengan sejumlah sensor nirkabel. Simpul sensor nirkabel mengumpulkan data dan mentransmisikannya ke *gateway* secara langsung atau, bila perlu, menggunakan *node* sensor nirkabel lainnya untuk meneruskan data ke *gateway*. Data yang dikirimkan kemudian dipresentasikan ke sistem melalui koneksi *gateway*. WSN terdiri dari perangkat independen yang tersebar secara spasial yang menggunakan sensor untuk

memantau kondisi fisik atau lingkungan. Perangkat independen ini, yang dikenal sebagai *router* dan *end node*, bersamaan dengan *gateway* bergabung untuk menciptakan sistem WSN. (Zaman, Ragaab, & Abdullah, 2012)

Ada beberapa topologi jaringan yang umum digunakan dalam membangun sebuah sistem WSN, yaitu:

1. Topologi *Star* topologi ini merupakan topologi paling dasar dimana setiap *node* mempertahankan satu jalur komunikasi langsung dengan *gateway*. Topologi ini sederhana namun membatasi jarak keseluruhan yang dapat dicapai.
2. Topologi *Cluster/Tree* arsitektur topologi *cluster* lebih komplek dibandingkan dengan topologi *star*. Setiap *node* masih mempertahankan satu jalur komunikasi untuk *gateway*. Perbedaannya menggunakan *node-node* lain dalam mengirimkan data, namun masih dalam satu jalur tersebut.
3. Topologi *Mesh* topologi ini merupakan solusi dari topologi-topologi sebelumnya, dengan menggunakan jalur komunikasi yang lebih banyak untuk meningkatkan kehandalan sistem. Dalam sebuah jaringan *mesh*, *node* mempertahankan jalur komunikasi untuk kembali ke *gateway*,



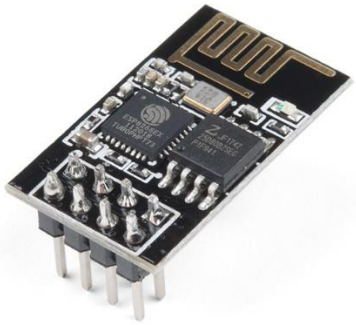
Gambar 2. 8 Topologi
Sumber: (Zaman, Ragaab, & Abdullah, 2012)

2.6 Esp8266-01

ESP8266 disebut sebagai *System On Chip (SOC)* yang memiliki kemampuan untuk terhubung dengan jaringan TCP/IP via Wi-Fi selain kemampuan layaknya mikrokontroler sebagai sebuah “otak” dan pengendali di dalam dunia elektronika. Modul ini dibuat oleh *Espressif System*, pabrikan asal cina.

Sebagai sebuah pendatang baru ditahun 2014, dengan produk pertamanya ESP-01. Produk ini langsung menyedot perhatian didunia elektronika karena kemampuannya yang memungkinkan mikrokontroler untuk terhubung dengan jaringan Wifi sesederhana menggunakan perintah *AT-Command*. Meskipun diawal kemunculannya, dokumentasi yang dibuat dalam bahasa cina, namun seiring berjalannya waktu, banyak pengguna yang sudah menerjemahkannya ke dalam berbagai bahasa sehingga penggunaannya tidak lagi sesulit dahulu.

Modul ini membutuhkan daya sekitar 3.3v dengan memiliki tiga mode wifi yaitu *station*, *access point* dan *both* (keduanya). Modul ini juga dilengkapi dengan prosesor, memori dan GPIO dimana jumlah *pin* bergantung dengan jenis ESP8266 yang kita gunakan. Sehingga modul ini bisa berdiri sendiri tanpa menggunakan mikrokontroler apapun karena sudah memiliki perlengkapan layaknya mikrokontroler. (Grokhotkov, 2018)



Gambar 2.9 ESP8266-01
Sumber: www.sparkfun.com

2.7 Sensor DHT11

Jenis tranduser yang digunakan untuk mengubah besaran mekanis, magnetis, panas, sinar, dan kimia menjadi tegangan dan arus listrik. Sensor sering digunakan untuk pendeteksian pada saat melakukan pengukuran atau pengendalian. Beberapa jenis sensor yang banyak digunakan dalam rangkaian elektronik antara lain *sensor* cahaya, sensor suhu.

Sensor suhu dan kelembapan (Sensor DHT11) merupakan sensor dengan kalibrasi sinyal digital yang mampu memberikan informasi suhu dan kelembapan. Sensor DHT11 dapat mengukur suhu dari 0-50 °C dengan persentasi kesalahan pembacaan yaitu 2°C dan kelembapan

20-90% (*Relative Humidity*) RH dengan persentasi kesalahan pembacaan sebesar 5% RH. Modul sensor yang digunakan dapat dilihat pada gambar.

Gambar 2.10 Sensor DHT11

Pada sensor DHT11 terdapat 4 *pin* yaitu VCC, Data, *Not Connected* dan *Ground*. Skema dari konfigurasi sensor DHT11 dapat dilihat pada gambar 2.10. (Kadir, 2015)

2.8 *Relay*

Relay adalah saklar tegangan listrik yang dioperasikan secara elektrik. *Relay* dapat diaktifkan atau dinonaktifkan menggunakan *pin input* pada *Relay* dengan menghubungkan ke *ground*.

Gambar 2.11 *Relay* 2 *channel*
Sumber: Perancangan

GND : dihubungkan dengan *ground*

IN1 : mengontrol *Relay* pertama dengan memberikan logika *LOW* atau menghubungkan dengan ground.

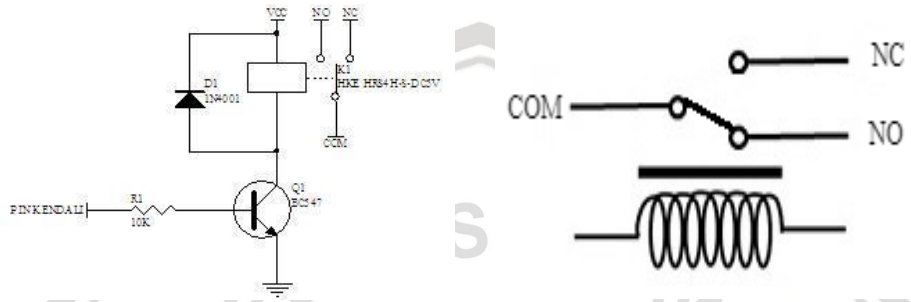
IN2 : mengontrol *Relay* kedua dengan memberikan logika *LOW* atau menghubungkan dengan *ground*.

VCC : input tegangan untuk mengaktifkan *Relay*, dihubungkan dengan 5V

COM : pin yang akan di koneksikan dengan pin NO atau NC.

NO (*Normally Open*): tidak ada kontak antara pin umum dan pin yang biasanya terbuka. Jadi, ketika Anda memicu *Relay*, itu terhubung ke pin COM dan pasokan disediakan untuk beban

NC (*Normally Close*): ada kontak antara pin umum dan pin yang biasanya tertutup. Selalu ada koneksi antara pin COM dan NC, bahkan ketika *relay* dimatikan. Ketika Anda memicu *relay*, rangkaian dibuka dan tidak ada pasokan yang disediakan untuk beban.



(a) (b)

Gambar 2.12 (a) skema *Relay* (b) sistem *Relay*
Sumber: (Saleh & Haryanti, 2017)

Gambar 2.12 memperlihatkan rangkaian dari *relay* serta sistem sakelar pada *relay*. Pada *relay* terdapat sebuah besi (*iron core*) yang dililit oleh kumparan *coil* yang berfungsi untuk mengendalikan besi tersebut. Apabila kumparan coil diberikan arus listrik, maka akan timbul gaya elektromagnetik yang kemudian menarik *armature* untuk berpindah dari posisi sebelumnya (NC) ke posisi baru (NO) sehingga menjadi saklar yang dapat menghantarkan arus listrik diposisi barunya (NO). Posisi dimana *armature* tersebut berada sebelumnya (NC) menjadi *open* atau tidak terhubung. Pada saat tidak dialiri arus listrik, *armature* akan kembali lagi ke posisi awal (NC). *Coil* yang digunakan oleh *relay* untuk menarik contact poin ke posisi close pada umumnya hanya membutuhkan arus listrik yang relatif kecil.

BAB III
METODE PENELITIAN

Dalam penelitian ini, dirancang suatu sistem *failover* pada *node Wireless Sensor Network* (WSN) menggunakan arduino untuk mengatasi *power loss*, sebagai pendukung pada penelitian ini. Adapun langkah – langkah yang dikerjakan adalah sebagai berikut.

1. Penetuan Ide Dasar
2. Perancangan dan Implementasi
3. Diagram Alir Perencanaan
4. Penarikan Kesimpulan

3.1 Penetuan Ide Dasar

Penentuan ide dasar adalah tahapan pertama untuk menyelesaikan penelitian ini. Metode *failover* yang didesain pada arduino adalah dengan menggunakan dua arduino dan satu ESP8266-01, dimana ketika arduino *master* kehilangan daya maka ada arduino *slave* yang mengambil alih fungsinya, sehingga pengiriman data dari *client* ke *server* tetap berjalan ketika terjadi kegagalan fungsi pada arduino *master* akibat dari pemeliharaan ataupun kehilangan daya. Setelah melakukan studi literatur dan pengecekan terhadap beberapa *syntax* pada beberapa bahasa pemrograman, akhirnya diperoleh ide dasar sebagai berikut.

- Menggunakan metode *failover* yang diterapkan pada mikrokontroler arduino, dimana setiap *node* terdapat dua arduino dan satu modul *wireless* (ESP8266).
- Membuat sistem node *Wireless Sensor Network* (WSN) yang terdiri dari *client* dan *access point* dengan data masukan dari sensor suhu dan kelempaman (DHT11).
- Sensor DHT11 digunakan secara bergantian pada sistem, ketika arduino *master* mati maka sensor DHT11 digunakan oleh arduino *slave*.
- Setiap *node* mendapatkan suplai daya dari catu daya 9V yang dihubungkan pada *power jack* arduino *master* dan arduino *slave*.
- *Powerbank* secara khusus hanya dihubungkan pada port USB arduino *slave*.
- Menggunakan rangkaian *internal* arduino untuk pemindahan pemakaian *power supply* pada arduino *slave* dari adaptor ke *powerbank.*

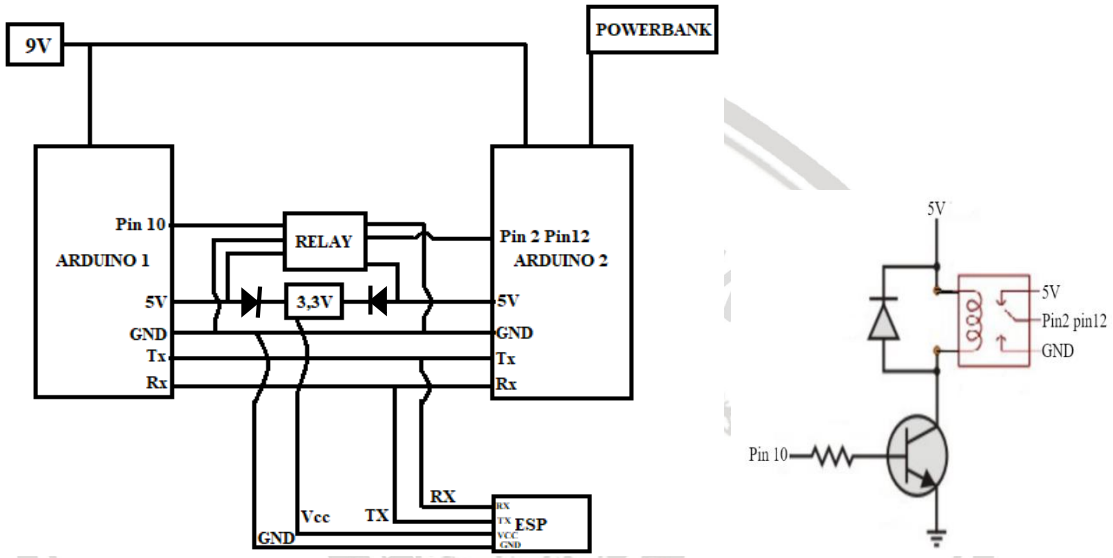
- Modul ESP8266 digunakan sebagai pengiriman data dan penerimaan data antar *node* secara *wireless*.
- Data suhu dan kelembapan pada *client* diproses pada arduino *master* dan arduino *slave* jika *master* tidak aktif.
- Pada *access point* modul ESP8266 mengirimkan data ke arduino *master* dan arduino *slave* jika *master* tidak aktif.
- Proses pengiriman data dari modul ESP8266 ke arduino (pada *access point*), dan pengiriman data dari arduino ke modul ESP8266 (pada *client*) menggunakan komunikasi *serial*.
- Pada arduino *slave* diberikan perintah *sleep*, sehingaga arduino *slave* tidak menerima atau mengirimkan data pada saat sistem berjalan.
- Menggunakan *relay* untuk memberikan *trigger LOW*, sehingga arduino cadangan *sleep* dan *trigger HIGH* untuk membangunkan arduino *slave* kembali.
- *Relay* yang digunakan untuk membangunkan arduino *salve* jika terjadi kegagalan pada arduino *master*, *input relay* dihubungkan pada arduino *master* dan *output Relay* NO (*Normally Open*) terhubung ke +5V pada arduino *slave*, NC (*Normally Close*) terhubung ke *ground*. Satu *pin output* dari *Relay* terhubung ke *pin* 2 dan *pin* 12. *Pin* 2 digunakan untuk mendeteksi perubahan sinyal dari *LOW* ke *HIGH* yang menunjukkan perubahan kondisi arduino *master*. *Pin* 12 memberikan informasi kepada arduino *slave* apakah harus dalam mode tidur atau mode bangun.

3.2 Perancangan

Pada perancangan dilakukan dari ide dasar yang telah dibuat untuk menerapkan skenario *failover*, digunakan dua Arduino Uno yang terhubung ke sensor *analog* dan satu *transceiver* ESP8266-01. Sirkuit ini juga dilengkapi dengan *Relay* untuk memberikan sinyal pemicu bangun kearduino *slave* dan regulator 3.3V sebagai catu daya untuk ESP. Secara khusus, Arduino *slave* menggunakan powerbank yang terhubung melalui *pin* USBVCC, *powerbank* dalam keadaan tidak difungsikan ketika perangkat mendapat sumber dari catu daya 9v. Pada perancangan *node* dibagi menjadi dua mode yaitu mode *access point* dan mode *client*.

- Gambar 3.1 menjelaskan rancangan sistem pada mode *acces point*. Pada sistem ini terdapat *Relay* yang *output*nya NO (*Normally Open*) terhubung ke *ground*, NC

(*Normally Close*) terhubung ke +5V. Satu *pin output* dari *Relay* terhubung ke *pin* 2 dan *pin* 12. *Pin* 2 digunakan untuk mendeteksi perubahan sinyal dari *LOW* ke *HIGH* yang menunjukkan perubahan kondisi arduino *master*. *Pin*12 memberikan informasi kepada arduino *slave* apakah harus dalam mode tidur atau mode bangun, sedangkan regulator AMS1117 digunakan untuk menurunkan tegangan menjadi 3,3 volt. Mode *access point* menunggu data yang dikirimkan *client*. Data diterima oleh modul ESP yang selanjutnya dikirimkan ke arduino untuk diproses.

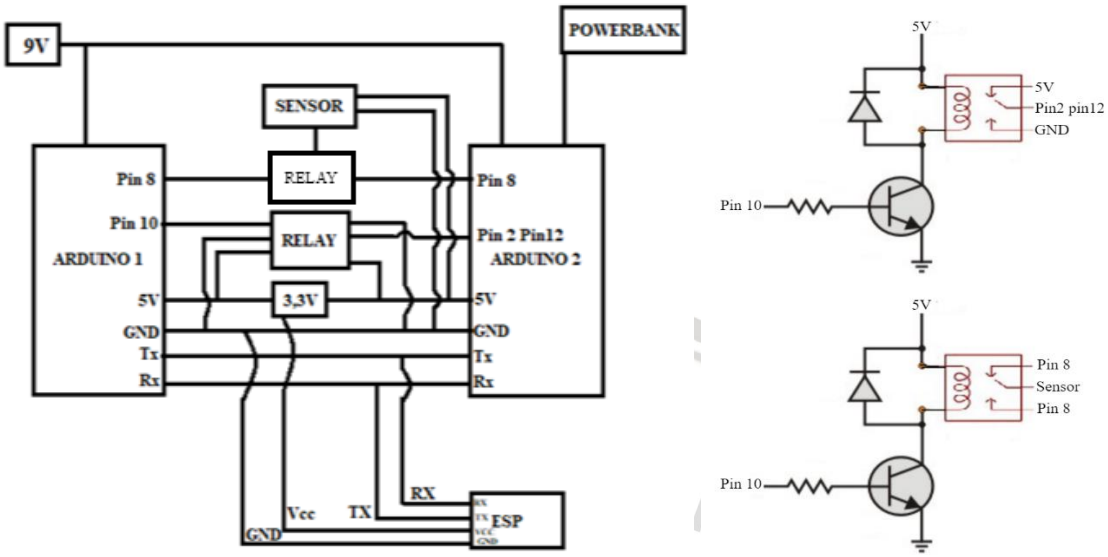


(a) (b)

Gambar 3.1 (a) Perancangan *node Access Point* (b) skema *Relay*
Sumber: perancangan

- Gambar 3.2 menjelaskan rancangan sistem pada mode *client*. Pada mode ini terdapat sensor suhu yang digunakan secara bergantian antara arduino *master* dengan arduino *slave*. Sensor suhu digunakan untuk mengetahui suatu kondisi dan datanya dikirimkan ke modul ESP melalui arduino seprti pada gambar 3.1. Pada mode *client* juga terdapat *Relay* yang digunakan untuk membangunkan arduino *slave* jika terjadi kegagalan pada arduino *master*, *output Relay* NO (*Normally Open*) terhubung ke *ground*, NC (*Normally Close*) terhubung ke +5V. Satu *pin output* dari *Relay* terhubung ke *pin* 2 dan *pin* 12. *Pin* 2 digunakan untuk mendeteksi perubahan sinyal dari *HIGH* ke *LOW* yang menunjukkan perubahan dalam kondisi arduino *master*. *Pin*12 memberikan informasi kepada arduino *slave* apakah harus dalam mode tidur atau mode bangun, serta satu buah *Relay* lagi

digunakan untuk pergantian penggunaan sensor suhu dan kelembapan. Pada mode *client* data yang diterima modul ESP akan diirimkan ke *access point* untuk diproses.



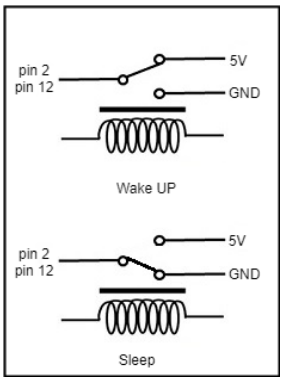
(a) (b)

Gambar 3.2 (a) Perancangan *node Client* (b) skema *Relay*
Sumber: perancangan

Setelah perancangan sistem selesai maka dilakukan pemrograman pada sistem *client* ataupun pemrograman pada sistem *access point*, pemrograman dilakukan terhadap arduino *master*, arduino *slave* dan ESP8266.

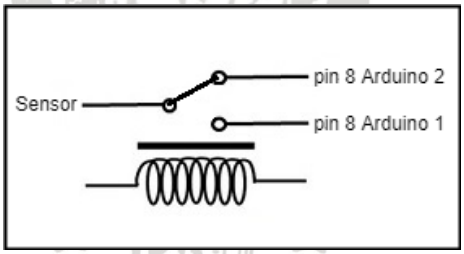
Pada mode *client* terdapat kondisi *sleep* pada arduino *slave* dan aktif jika terjadi kegagalan pada arduino *master* akibat dari pemeliharaan terhadap arduino *master* atau kehilangan daya, ketika arduino *slave* bangun dan melakukan *takeover* sehingga modul ESP8266 menerima data sensor dari arduino *slave*, dan dikirimkan ke *access point*.

Pada sistem yang terdapat pada mode *access point* juga terdapat kondisi *sleep* pada arduino *slave*, sehingga jika terjadi kegagalan pada arduino *master* akibat dari pemeliharaan terhadap arduino *master* atau ke hilangan daya listrik maka arduino *slave* mengetahui kondisi pada arduino *master* akibat perpindahan dari *LOW* ke *HIGH* pada *output relay* sehingga arduino *slave* bangun dan melakukan takeover atau pengambilan alih fungsi dari arduino *master*, sehingga proses pada *node* tidak terjadi kehilangan data.



Gambar 3.3 *Relay* mode wake up dan sleep
Sumber: perancangan

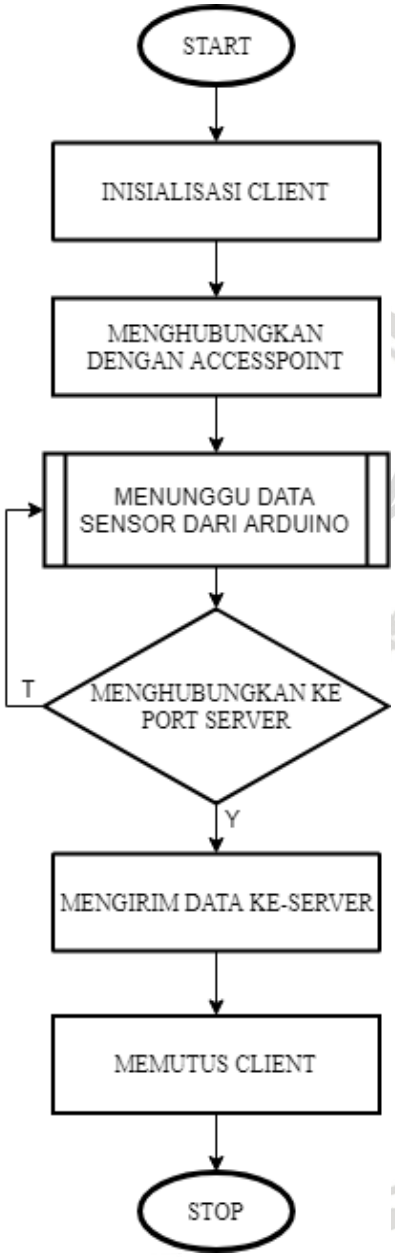
Gambar 3.3 menunjukan penggunaan *relay* pada sistem, sehingga arduino cadangan dapat dalam mode *sleep* atau mode *wakeup*. *Pin* 2 digunakan sebagai *interupsi* untuk membangunkan arduino *slave* dan *pin* 12 digunakan untuk memberikan kondisi *sleep*. *Pin* 2 dan *pin* 12 dihubungkan pada COM *Relay*, +5V dihubungkan pada *normaly close relay* serta *pin ground* pada *normaly open relay*. Ketika *relay* aktif maka pin 2 dan pin 12 akan tehubung ke ground, sehingga ketika *pin* 12 mendapatkan logika *LOW* maka arduino *slave* akan dalam mode *sleep*, ketika *relay* tidak aktif maka *pin* 2 dan *pin* 12 akan tehubung dengan +5v sehingga *pin* 2 mendapatkan logika *HIGH* maka arduino cadangan akan dalam mode *wake up*.



Gambar 3.4 *Relay* perpindahan sensor
Sumber: perancangan

Gambar 3.4 menunjukan pemakaian sensor bersama menggunakan *relay*, yang mana r*elay* digunakan untuk perpindahan pemakaian sensor arduino *slave* ke arduino *master* ketika r*elay* aktif, dan ketika *relay* mati maka sensor akan digunakan arduino *slave*. Pin sensor dihubungkan pada com *relay* dan *normaly close* dihubungkan pada *pin* 8 arduino *slave* serta *normaly open relay* dihubungkan pada *pin* 8 arduino *master*. Dari perancangan maka dibuat program yang mendukung sistem yang telah dibuat.

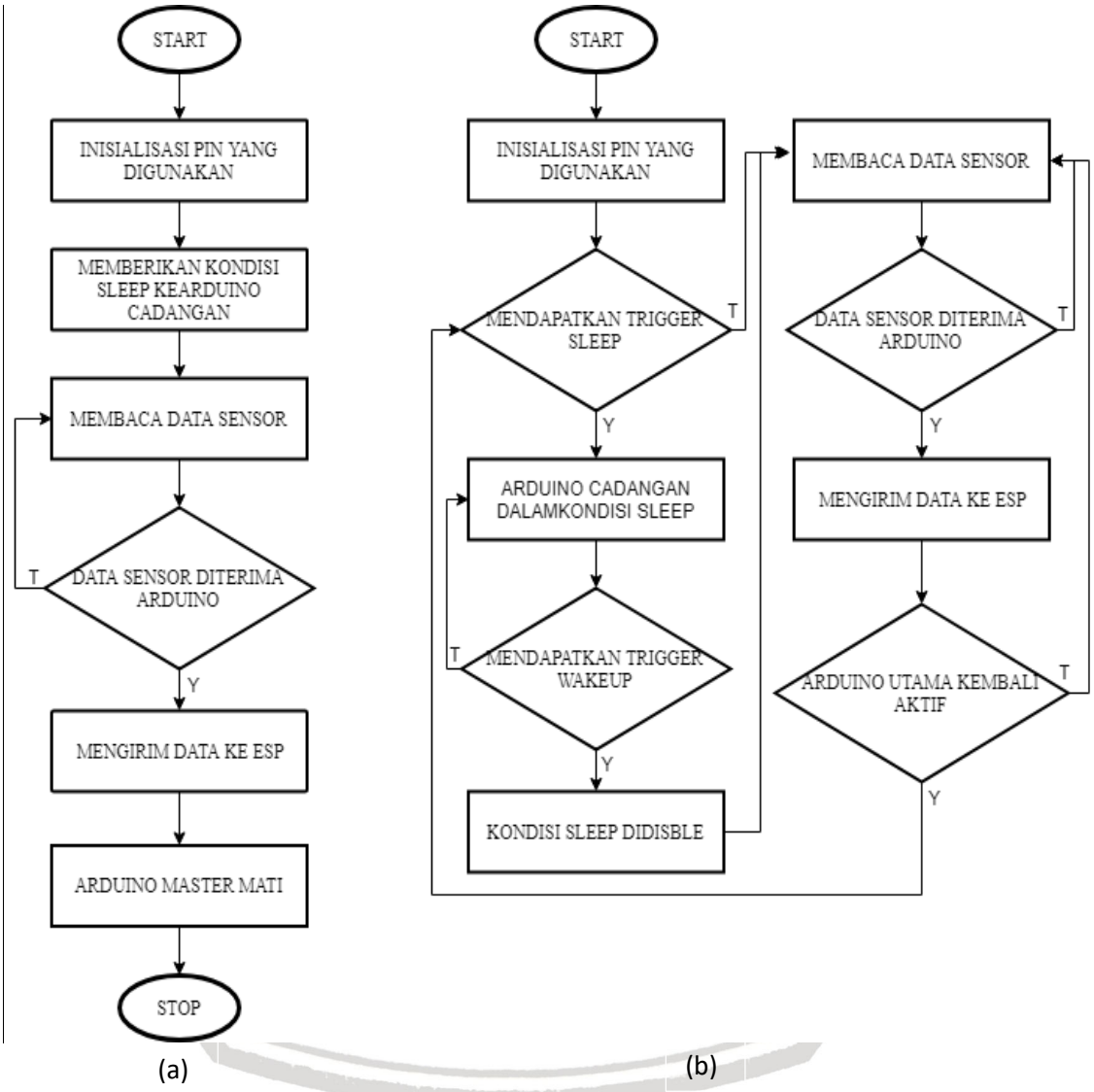
Pemrograman arduino *master*, pemrograman arduino *slave*, dan pemrograman ESP8266 pada *node client* dan node *access point* digambarkan dalam diagram alir program sebagai berikut :



Gambar 3.5 Diagram alir pemrograman ESP *client*
Sumber: perancangan

Gambar 3.5 menunjukan pemrograman pada *client* yang diprogram kedalam ESP8266. ESP8266 terhubung dengan *access point* berikutnya setelah *access point* terhubung, ESP

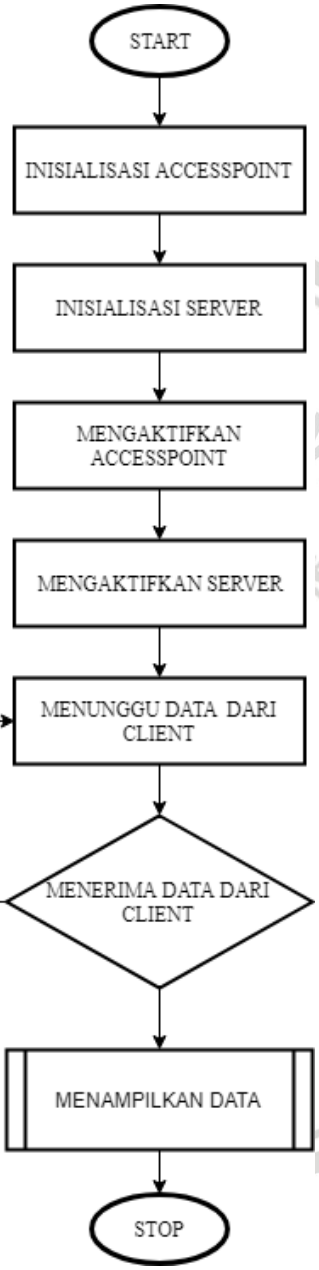
menunggu data yang dikirimkan arduino. Ketika ESP telah menerima data dari arduino maka ESP menghubungkan ke *port server*. Setelah terhubung ke *port server* maka ESP mengirimkan data yang sudah diterima ke mode *access point*, setelah data selesai dikirim maka *client* diputus dan memulai menunggu data sensor.



Gambar 3.6 Diagram alir pemrograman (a) arduino *master* (b) arduino *slave*
Sumber: perancangan

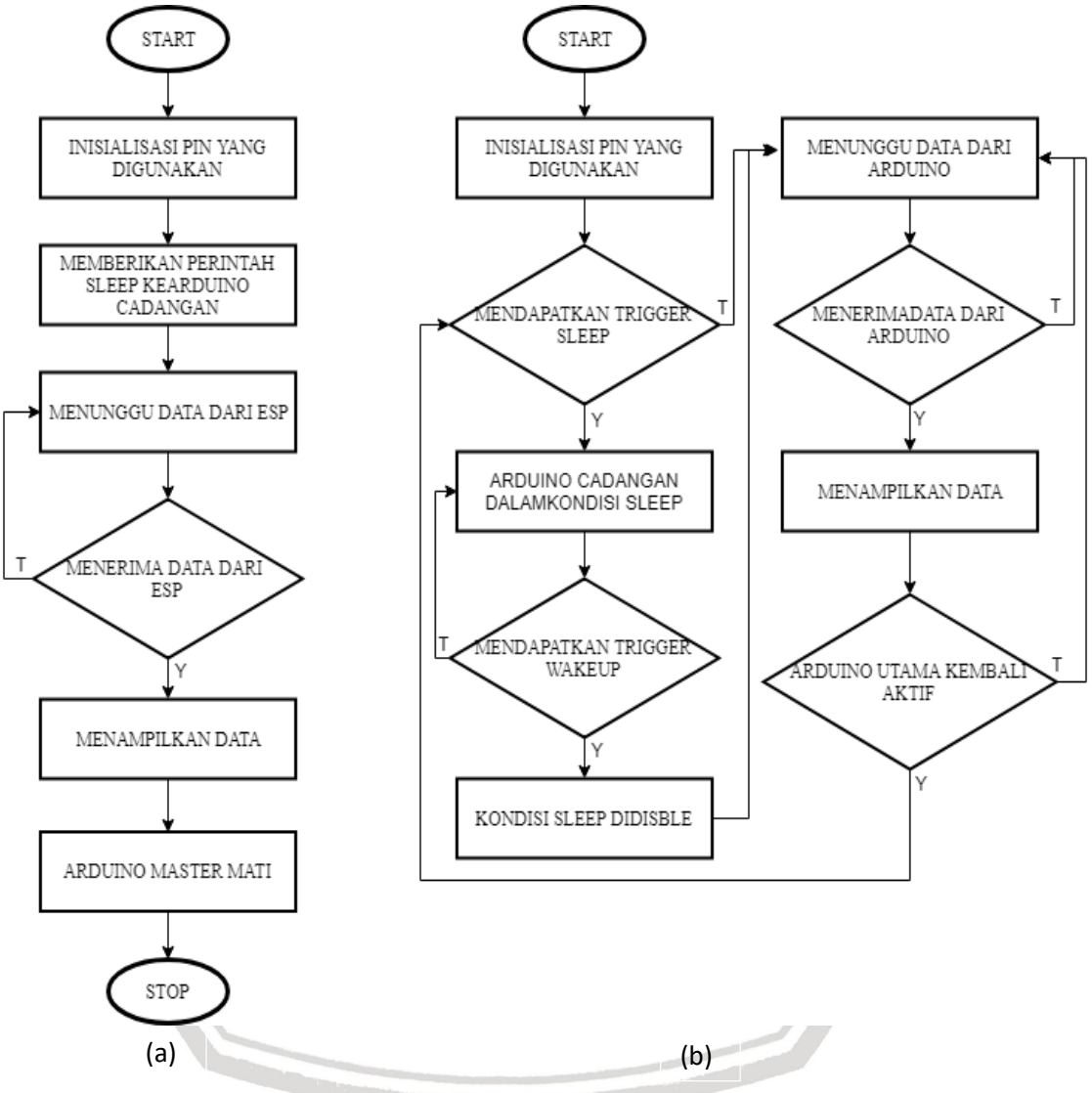
Gambar 3.6 menunjukan diagram alir pemrograman pada *client* yang diprogram pada arduino. Arduino membaca data yang dikirimkan oleh sensor setelah data diterima maka arduino

mengirimkan data sensor kepada ESP. Gambar 3.6 (a) memperlihatkan pemrograman pada arduino *master* dan gambar 3.6 (b) memperlihatkan pemrograman pada arduino *slave*. Pada arduino *slave* diterapkan program mode *sleep* sehingga arduino *slave* tidur dan dalam kondisi aktif ketika mendapat *trigger* dari arduino utama.



Gambar 3.7 Diagram alir pemrograman ESP *access point*
Sumber: perancangan

Gambar 3.7 menunjukan pemrograman pada mode *access point* yang diprogram ke dalam ESP8266. Pada mode ini ESP8266 mengaktifkan *access point* dan mengaktifkan *server*, setelah *server* aktif mode *acces point* menunggu data dari *client*, setelah *access point* menerima data dari *client* maka modul ESP8266 mengirmkan data yang diterima ke arduino untuk ditampilkan pada *serial com* arduino.



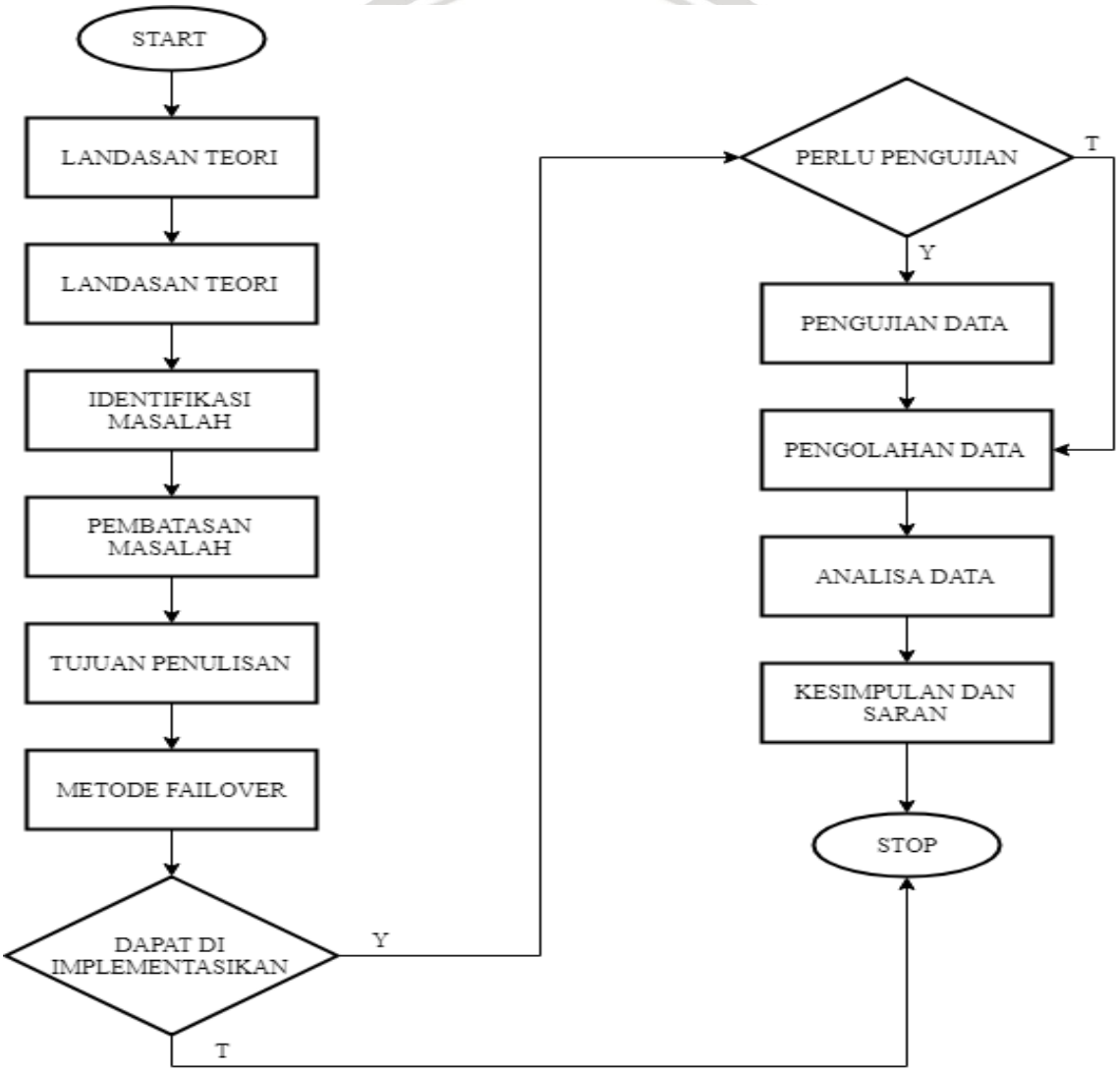
Gambar 3.8 Diagram alir pemrograman (a) arduino *master* (b) arduino *slave access point*
Sumber: perancangan

Gambar 3.8 menunjukan diagram alir pemrograman pada *access point* yang diprogram pada arduino. Arduino membaca data yang dikirimkan oleh modul ESP dan ditampilkan pada *serial monitor* arduino. Gambar 3.8 (a) meperlihatkan diagram alir pemrograman pada arduino *master*

dan pada gambar 3.8 (b) memperlihatkan diagram alir pemrograman pada arduino *slave*, yang mana pada arduino *slave* diterapkan mode *sleep* sehingga arduino cadangan tidur dan dalam mode aktif ketika mendapat *trigger* dari arduino utama.

3.3 Diagram Alir Perencanaan

Analisis metode *failover* pada *node Wireless Sensor Network* (WSN) menggunakan arduino dan modul ESP8266 dirancang diagram alir perencanaan sebagai fitur pendukung pada sistem *Wireless Sensor Network* (WSN) ini terdiri dari beberapa tahap yang digambarkan sebagai berikut :



Gambar 3.9 Flowchart Perencanaan Metode *Failover*
Sumber: perancangan

3.4 Pengujian dan analisis

Menjelaskan langkah-langkah pengujian dari sistem yang telah di buat dan membahas hasil pengujiannya. Pengujian metode *failover* diterapkan pada *client* dan *access point*.

- Pengujian terhadap sistem *failover* untuk membuktikan apakah sistem yang telah dirancang dapat melakukan *backup* atau tidak dan melihat *recovery time* pada saat sistem melakukan *backup* yang diskenariokan dengan mematikan arduino utama atau mencabut adaptor *power supply* arduino.
- Pengujian *failback* sistem untuk melihat apakah sistem dapat kembali ke proses awal ketika arduino utama diaktifkan kembali atau menghubungkan kembali *power supply*.

3.5 Penarikan Kesimpulan

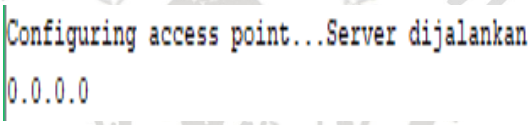
Pada tahap ini, kesimpulan dan analisis dari pengujian diuraikan. Tahap selanjutnya adalah pembuatan saran untuk perbaikan data dari sistem yang telah dibuat.

BAB IV
PENGUJIAN DAN ANALISIS

Untuk mengetahui suatu sistem yang sudah dirancang bekerja dengan baik dan sesuai dengan perancangan, maka diperlukan serangkaian pengujian. Pengujian yang dilakukan dalam penelitian ini adalah sebagai berikut:

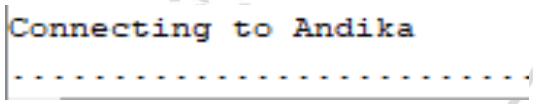
1. Pengujian Metode *failover*
2. Pengujian *failback Sistem*

Memulai pengujian dengan mengaktifkan sistem yang telah dirancang pada *node access point* dan *node client*. Pengakatifan sistem dimulai dengan *node access point* kemudian pengaktifan pada *node client*. Pengaktifan pada setiap *node* dapat dimulai dengan *master*, *slave* ataupun keduanya. Pada pengujian ini pengaktifan dilakukan dengan mengaktifkan kedua arduino secara bersamaan.



Gambar 4.1 Pengaktifan Mode *Access Point*
Sumber: Perancangan

Gambar 4.1 memperlihatkan pengaktifan mode *access point* dan mengaktifkan *server* setelah diaktifkan maka ditampilkan pada *serial com* arduino. Setelah *access point server* berhasil diaktifkan maka selanjutnya mengaktifkan *client*. Mengaktifkan *client* arduino *master* ataupun *slave* diaktifkan bersamaan. Ketika arduino aktif, ESP menghubungkan *client* ke *access point*. Pengaktifkan client dapat dilihat pada Gambar 4.2.



Gambar 4.2 Pengaktifan Mode *Client*
Sumber: Perancangan

Pada pengaktifan arduino secara bersamaan, arduino *master* mengaktifkan *relay* dan memberikan logika LOW pada arduino *slave*, ketika arduino *slave* membaca nilai *low* yang diberikan maka arduino *slave* berada pada mode *sleep* dan arduino *master* mengirim data (pada mode *client*) dan menerima data (pada mode *acces point*). Ketika sistem telah siap maka

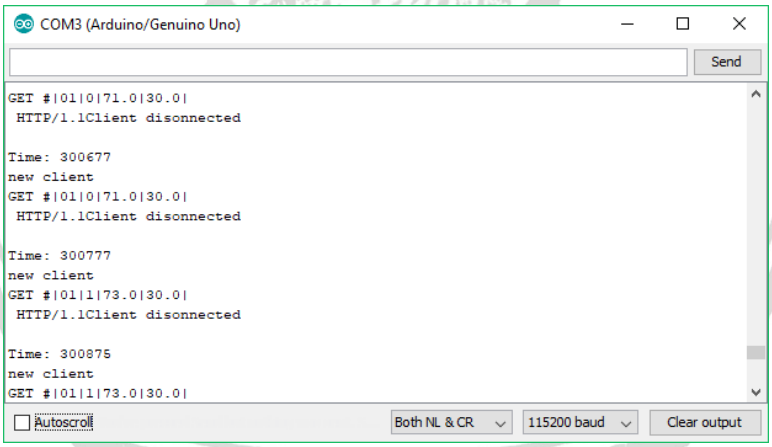
dilakukan sekenario *failover*. Sekenario *failover* dilakukan secara bergantian pada *node acces point* dan *node client*. Setelah sekenario *failover* dilakukan maka selanjutnya melakukan *failback* pada sistem dengan menyalakan kembali arduino *master*.

4.1 Pengujian Sistem *Failover*

Pengujian terhadap sistem *failover* untuk membuktikan apakah sistem yang telah dirancang dapat melakukan *backup* atau tidak dan melihat apakah sistem kehilangan data saat terjadi kegagalan pada arduino *master*. Data sekenario *failover* pada *client* diambil dengan melihat waktu antara arduino *master* mengirimkan atau menerima data terakhir dan waktu arduino *slave* mengirimkan atau menerima data pertama, sedangkan sekenario *failover* pada *node acces point* data diambil ketika *client* terjadi putus koneksi. Untuk mempermudah pengujian maka diberikan perintah *millis ()*, sehingga waktu dapat langsung dilihat pada *serial monitor* arduino.

4.1.1 Pengujian pada *Client*

Pengujian pada *client* dilakukan dengan menerapkan sekenerio *failover* hanya pada *node client*, sedangkan pada *node access point* dibuat tetap dengan menghubungkan pada komputer, sehingga data yang diterima *node accespoint* ditampilkan pada *serial com* arduino. Data pengujian sekenario *failover* yang diterapkan pada *client* diambil dari *node access point* yang ditampilkan pada *serial com* arduino. Data pengujian *failover* pada *client* dapat dilihat pada gambar sebagai berikut:



Gambar 4.3 Pengujian *failover* pada mode *client*
Sumber: perancangan

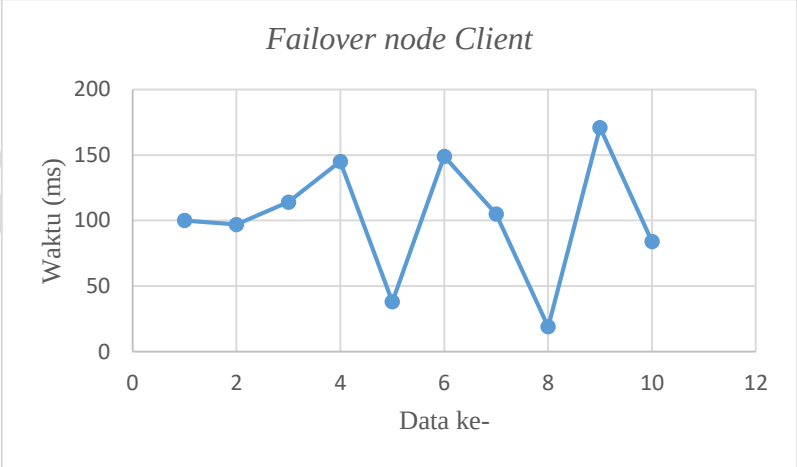
Gambar 4.3 menunjukkan contoh *failover* pada data pertama. Data *master* di tunjukan dengan (|0|) dan data *slave* ditunjukan dengan (|1|) , pada gambar diatas *master* mati pada saat waktu mencapai 300677 ms dan *slave* menggantikan sistem pada saat 300777 ms, data yang diambil adalah waktu pergantian antara *master* ke *slave* sehingga didapat 100 ms. Pada

pengujian *failover* pada *client* data diambil sebanyak 10 kali pengujian sehingga didapat sesui ditunjukan pada tabel.

Tabel 4.1 Hasil pengujian mode *client*
Sumber: perancangan

no	kejadian	Waktu arduino *master* mati (ms)	Waktu arduino *slave* hidup (ms)	*Failover* (ms)
1	Kehilangan daya	300677	300777	100
2	Kehilangan daya	800816	800913	97
3	Kehilangan daya	1503664	1503778	114
4	Kehilangan daya	2100697	2100842	145
5	Kehilangan daya	2701391	2701429	38
6	Kehilangan daya	3302638	3302787	149
7	Kehilangan daya	3902039	3902144	105
8	Kehilangan daya	4400406	4400425	19
9	Kehilangan daya	5000590	5000761	171
10	Kehilangan daya	5599737	5599821	84
Rata - rata		2961266	2961368	102.2

Berdasarkan hasil pengujian pada tabel 4.1 Penerapan metode *failover* pada *client* berhasil meminimalisir kehilangan data yang dikirimkan *client* akibat dari perawatan pada arduino utama ataupun sistem mengalami kehilangan daya. Pada tabel 4.1 Pengujian ini dilakukan dengan rata-rata waktu kegagalan sistem 2961266 ms dan rata-rata waktu *slave* mengambil alih sistem 2961368 ms sehingga rata-rata *failover* sistem adalah 102.2 ms. Pada sistem ini *client* mengirimkan data setiap 100 ms atau setiap detik *node client* mengirimkan 10 data. Sehingga sistem mengalami kehilangan data rata-rata sebanyak 2 data.

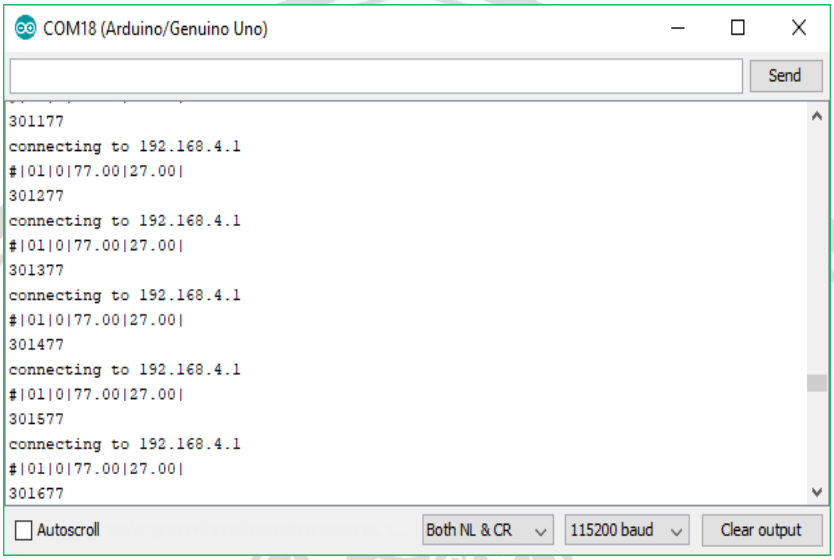


Gambar 4.4 Grafik pengujian *failover* node *client*
Sumber: perancangan

Gambar 4.4 menunjukan grafik *failover* pada *node client. Failover* minimum terdapat pada data ke 8 dengan kecepatan 19 ms dan *failover* maksimum pada data ke 6 dengan kecepatan 149 ms. Grafik diatas menggambarkan bagaimana arduino *slave* telah siap mengambil alih sistem ketika menjadi mode bangun pada saat arduino *master* kehilangan daya.

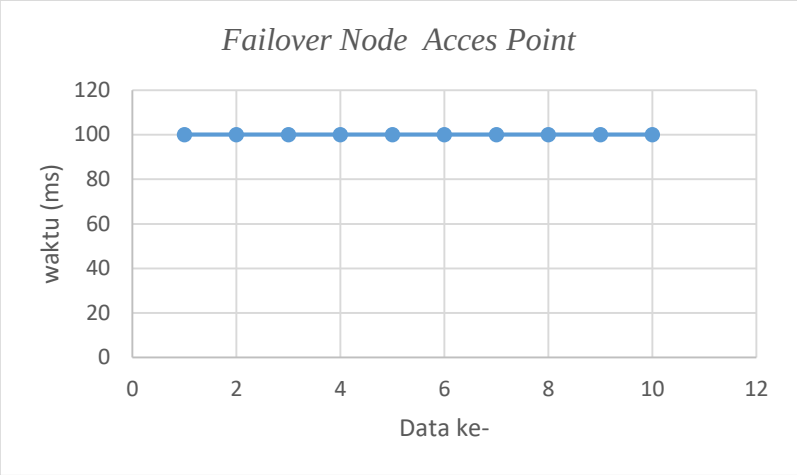
4.1.2 Pengujian pada *Access Point*

Pengujian *failover* pada *acces point* dilakukan dengan membuat sekenario *failover* pada *acces point* dan *client* dihubungkan pada komputer dan menampilkan pengiriman data pada *serial com* arduino. Data *failover* pada mode *access point* diambil dari *client* ketika *client* mengalami gagal koneksi. Data pengujian *access point* dapat di lihat pada Gambar 4.5.



Gambar 4.5 Hasil pengujian *failover* pada mode *acces point*
Sumber: perancangan

Gambar 4.5 menunjukkan contoh *failover* pada *access point* dimana *client* mengirimkan data setiap 100 ms atau mengirimkan 10 data setiap detik. Pada gambar diatas *client* tidak mengalami gagal koneksi ketika *node acces point* diterapkan sekenario *failover*. Pada 10 kali pengujian *failover* pada *access point* didapatkan hasil sesuai dengan Gambar 4.6.



Gambar 4.6 Grafik hasil pengujian pada mode *Access Point*
Sumber: perancangan

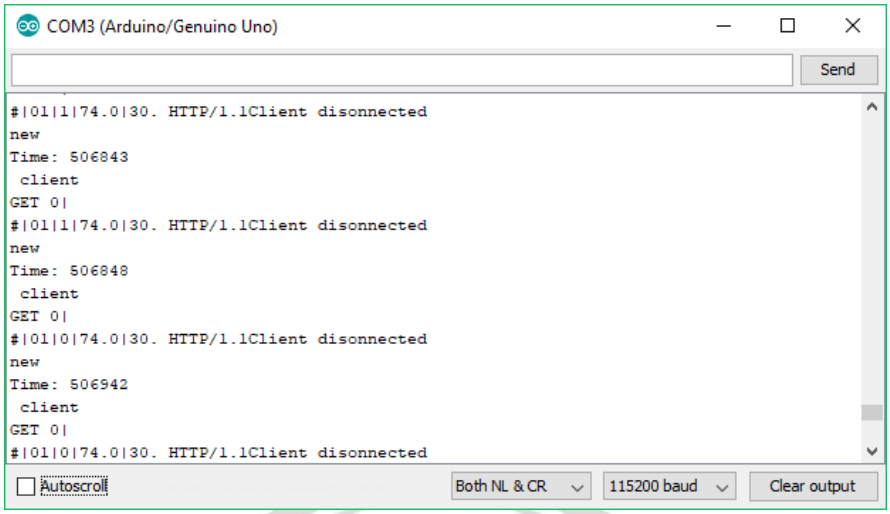
Gambar 4.6 Menunjukkan hasil dari pengiriman node *client* ke *acces point*. Berdasarkan grafik hasil pengujian *failover* pada *access point* dengan 10 hasil pengujian didapatkan kegagalan rata-rata 0 ms, keberhasilan dari kongfigurasi mode *access point* dalam menerima data dikerenakan pada kongfigurasi ini arduino hanya digunakan sebagai penerima data dari modul ESP8266-01 sehingga modul ESP8266-01 tetap aktif ketika sekenario *failover* diterapkan.

4.2 Pengujian *Failback*

Pengujian *failback* sistem dilakukan untuk melihat apakah sistem dapat kembali ke proses awal ketika arduino utama diaktifkan kembali atau menghubungkan kembali *power supply*. Pengujian *failback* dilakukan dengan melihat waktu perpindahan antara arduino *master* dan arduino *slave* (pengujian pada *client*) dan melihat apakah terjadi putus koneksi ketika *failback* dilakukan (pengujian pada *acces point*).

4.2.1 Pengujian pada *Client*

Failback dari metode *failover* pada mode *client* dilakukan dengan mengaktifkan arduino *master*. Data pengujian *failback* diambil ketika arduino *master* mengirimkan data pertama dan arduino *slave* mengakhiri pengiriman dan masuk ke mode *sleep*. Pengujian pada *client* dapat dilihat pada Gambar 4.7.



Gambar 4. 7 Hasil pengujian *failback* pada *client*
Sumber: perancangan

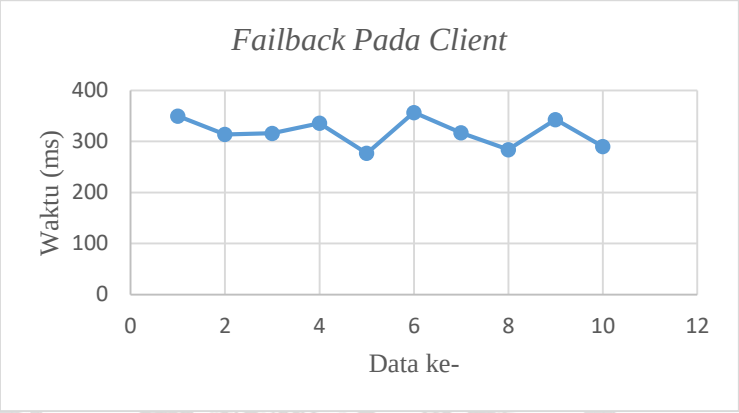
Gambar 4.7 menunjukkan *failback* sistem pada mode *client.* Data pengujian *failback* diambil pada *node access point* yang dikirimkan pada *serial monitor* arduino. Pada pengujian *failback*, data didapat ketika arduino *slave* masuk pada mode tidur dengan waktu 536084 ms dan arduino *master* mengirimkan data pertama dengan waktu 536434 ms sehingga didapat waktu *failback* dari sistem 350 ms. Pengujian *failback* sistem dilakukan 10 kali pengujian sesuai dengan tabel 4.2.

Tabel 4.2 Hasil pengujian *failback* mode *client*
Sumber: perancangan

no	kejadian	Waktu arduino *master* mati (ms)	Waktu arduino *slave* hidup (ms)	*Failback* (ms)
1	Pengembalian Sistem	536084	536434	350
2	Pengembalian Sistem	1206436	1206750	314
3	Pengembalian Sistem	1806453	1806769	316
4	Pengembalian Sistem	2444532	2444868	336
5	Pengembalian Sistem	3024541	3024818	277
6	Pengembalian Sistem	3682562	3682919	357
7	Pengembalian Sistem	4106981	4107298	317
8	Pengembalian Sistem	4702340	4702624	284
9	Pengembalian Sistem	5296098	5296441	343
10	Pengembalian Sistem	5904344	5904634	290
Rata - rata		3271027	3271356	318.4

Berdasarkan hasil pengujian pada tabel 4.2 pengujian *failback* pada *client* berhasil meminimalisir kehilangan data yang dikirimkan *client* akibat dari pergantian pada arduino *salve*

yang memasuki mode *sleep* ke arduino utama yang telah aktif kembali. Pada table 4.2 Pengujian ini dilakukan dengan rata-rata waktu ketika arduino *slave* memasuki mode *sleep* adalah 3271027 ms dan rata-rata waktu arduino *master* aktif kembali dan mengambil alih sistem 3271356 ms sehingga rata-rata *failback* sistem adalah 318.4 ms. Pada pengujian ini arduino *master* memulai sistem dari awal sehingga pengembalian sistem lebih lama dari *failover* dan memberikan kondisi *sleep* ketika arduino *master* telah siap mengirimkan data. Sehingga sistem mengalami kehilangan data rata-rata sebanyak 4 data.



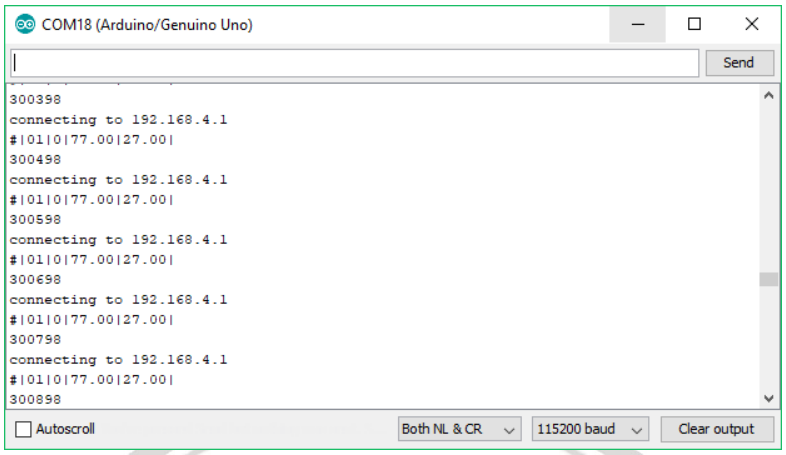
Gambar 4.8 Grafik hasil pengujian *failback* pada *client*
Sumber: perancangan

Gambar 4.8 memperlihatkan grafik pengujian *failback* dari metode *failover*, *client* berhasil mengembalikan sistem seperti semula setelah terjadinya *failover* pada *client* akibat dari perawatan pada arduino *master* ataupun sistem mengalami kehilangan daya. Dari 10 kali pengujian didapatkan hasil pengujian pengembalian sistem atau *failback* didapatkan waktu pengembalian sistem dengan waktu minimum 277 ms dan pengambalian mkasimum 357 ms, grafik diatas menggambarkan waktu pengembalian ke sistem utama dimana sistem utama memulai dari awal atau *restart* dikarenakan kehilangan daya sehingga *failback* sistem setelah terjadi *failover* lebih lama dibandingakn dengan *failover* sistem yang mana arduino *slave* telah siap ketika arduino master mati.

4.2.2 Pengujian pada *Access Point*

Pengujian failback dari metode *failover* pada mode *accesspoint* dilakukan dengan mengaktifkan arduino *master* dan melihat pengiriman pada *client* yang ditampilkan pada *serial com* arduino. Data pengujian *failback* diambil pada *client* ketika arduino utama diaktifkan

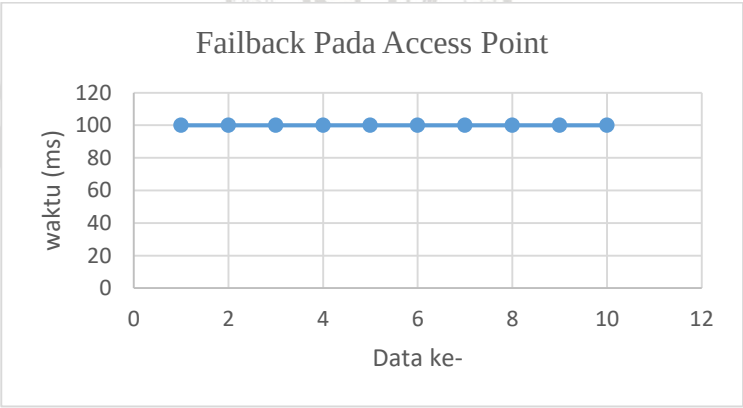
apakah terdapat putus koneksi ke *server* ketika pergantian arduino *master* ke arduino *slave*, pengujian *failback* pada *accespoint* dapat dilihat pada gambar 4.9.



Gambar 4.9 Hasil pengujian *failback* pada *access point*
Sumber: perancangan

Gambar 4.9. menunjukkan *failback* pada sistem *acces point* yang ditampilkan *client* pada *serial com* arduino. Pengiriman yang dilakukan setiap 100 ms atau setiap detik mengirimkan 10 data. Pada Gambar 4.9 diperlihatkan pengiriman *client* tidak mengalami putus koneksi pada saat terjadi *failback* pada *node access point*.

Failback dari metode *failover* pada konfigurasi sistem yang telah dirancang pada mode *access point* didapatkan hasil dari 10 pengujian dengan rata-rata *failback* pada kongfigurasi sistem adalah 0 ms. Keberhasilan dari konfigurasi mode *access point* dikarenakan pada konfigurasi ini arduino hanya digunakan sebagai penerima data dari ESP dan pada sistem tidak terjadi kehilangan data.



Gambar 4.10 Grafik hasil pengujian *failback* pada *access point*
Sumber: perancangan

Gambar 4.10 memperlihatkan pengujian *failback* dari metode *failover* yang diterapkan pada mode *access point*. Mode *access point* berhasil mengembalikan sistem seperti semula setelah terjadinya *failover* dan tidak terjadi kehilangan data. Pada 10 kali hasil pengujian yang dilakukan didapatkan 0 kehilangan data saat terjadi *failback*, dikarenakan *client* selalu terhubung dengan *node acces point*. Pada *access point* modul ESP8266-01 selalu aktif dan arduino hanya sebagai penerima data.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut:

1. Sistem *failover* dapat mengatasi kehilangan daya pada *node Wireless Sensor Network* dengan menggunakan mekanisme prosesor *dual* redundansi yang menggunakan sistem utama dan sistem cadangan.
2. Sistem yang dibangun dapat bekerja dengan baik sesuai dengan konsep kerjanya, sehingga *client* dan *access point* dapat mengirim dan menerima data ketika terjadi pemeliharaan pada arduino *master* atau kehilangan pasokan daya utama. *Failover* pada *client* dari 10 data pengujian, didapatkan rata-rata kecepatan sistem *failover* dengan waktu 102,2 ms serta waktu minimum 19 ms dan waktu maksimum 149 ms dan rata-rata kehilangan 2 data dikarenakan arduino *slave* telah siap mengirim data ketika mode bangun. *Failover* pada *access point* dari sepuluh pengujian didapatkan *failover* dengan waktu 0 ms dikarenakan modul ESP8266 selalu aktif menerima data.
3. *Failback* sistem setelah terjadinya *failover* didapatkan dari 10 pengujian pada *node client* dengan kecepatan minimum 277 ms dan kecepatan maksimum 357 ms serta kehilangan 4 data. Hal ini terjadi dikarenakan arduino utama memulai kembali sistem atau *restart. Failback* pada *access point* dari sepuluh pengujian didapatkan *failback* dengan waktu 0 ms dikarenakan modul ESP8266 selalu aktif menerima data.
4. Sistem *failover* yang dibangun pada skripsi ini menggunakan komunikasi *serial* sehingga sistem dapat berjalan dan tidak ada kendala disaat salah satu sistem mati. Arduino membaca data dan mengirimkan ke ESP proses pada *client* dan ESP mengirimkan data pada arduino proses pada *access point*.

5.2 Saran

1. Berdasarkan penelitian yang telah dilakukan disarankan pada penelitian selanjutnya dapat menggunakan modul wifi yang lainya seperti X-BEE.
2. Selain metode *failover*, metode *load balanc*ing bisa menjadi solusi dengan terlebih dahulu dilakukan penelitian untuk hal tersebut.

DAFTAR PUSTAKA

Abrams, I. V. (2014, Maret 14). *What happens if I power the Arduino with both the USB and external power voltage*. Retrieved from stackexchange.com: https://arduino.stackexchange.com/questions/893/what-happens-if-i-power-the-arduino-with-both-the-usb-and-external-power-voltage

Arduino. (2018). *arduino uno rev3*. Retrieved from arduino.cc: https://store.arduino.cc/usa/arduino-uno-rev3

Artanto, D. (2012). *Interaksi Arduino dan Labview.* Jakarta: PT. Elex Media Komputindo.

Dicola, T. (2018). *Low Power WiFi Datalogger*. Retrieved from Learn.adafruit.com: https://learn.adafruit.com/low-power-wifi-datalogging/power-down-sleep

Grokhotkov, I. (2018). *ESP8266 Arduino Core Documentation*. Retrieved from esploradores.com: http://www.esploradores.com/wp-content/uploads/2018/06/arduino-esp8266.pdf

Kadir, A. (2015). *Arduino.* Yogyakarta: Penerbit Andi.

Mulyanto, U. (2016). Implementasi Disaster Recovery Plan Server System dengan Metode Failover dan Miroring CO-Location Server Berbasis Linux Di Fakultas Kedokteran Unissula. *Universitas STIKUBANK Semarang*.

Purnomo, N., & Syafrizal, M. (2013). Failover Cluster Server dan Tunneling Eoip untuk Sistem Disaster Recovery. *AMIKOM Yogyakarta*.

Rohani, H., & Roosta, K. (2014). Calculating Total Sistem Availability. *semanticscholar.org*.

Saleh, M., & Haryanti, M. (2017). Rancang Bangun Sistem Keamanan Rumah Menggunakan Relay. *Universitas Mercu Buana*.

Stallings, W. (2007). *Data and computer communications 8th.* new jersey: Pearsoon Education .inc.

Wahyuno, T. (2003). *Prinsip Dasar dan Teknologi Komunikasi Data.* Yogyakarta: Graha ilmu.

Zaman, N., Ragaab, K., & Abdullah, A. b. (2012). Wireless Sensor Network and Energy Efficiency Protokol, Routing and Management. *IGI global*.

Lampiran 1

Program ESP pada *Access Point*

```
#include <ESP8266WiFi.h>

const char* ssid = "Andika";
// Nama AP/Hotspot

const char* password = "1234567890";  // Password AP/Hotspot

char H [80];

WiFiServer server(80);

void setup() {

  Serial.begin(115200);

  delay(10);

 // Mengatur WiFi ----------
---------------------------------
------------

  Serial.println();

  Serial.print("Configurating access point...");

  WiFi.mode(WIFI_AP);
// Mode AP/Hotspot

  WiFi.softAP(ssid, password);

 // Start the server -----------
---------------------------------
------------

  server.begin();

  Serial.println("Server dijalankan");

 // Print the IP address -------
---------------------------------
------------
Serial.println(WiFi.localIP());

 }

void loop() {

  // Check if a client has connected

  WiFiClient client = server.available();

  if (!client) {

    return;

    }

  // Wait until the client sends some data

  Serial.println("new client");

  while(!client.available()){

    delay(1);

    }

  // Read the first line of the request

  String req = client.readStringUntil('\r');

  req.toCharArray(H,50);

  Serial.write (H);

  client.flush();

  Serial.println("Client disonnected");

}
```

Lampiran 2

Program pada Arduino *Master*

```
char r [100];
unsigned long time;
int Relay = 7;
void setup() {
  pinMode(Relay, OUTPUT);
  Serial.begin(115200);
  Serial.println("Is this on?...");
  digitalWrite(Relay, LOW);
}
void loop() {
  if(Serial.available() > 0) {
    Serial.readBytes(r,63);
    Serial.print("Time: ");
    time = millis();
    Serial.println(time);
    Serial.println(r);
  }
 }
```

Lampiran 3

Program pada Arduino *Slave*

```
#include <avr/sleep.h>


char r [100];

unsigned long time;

int ledPin = 13;
// LED connected to digital pin 13

int sleepPin = 12;
// active LOW, ground this pin
momentary to sleep

int wakePin = 2;
// active LOW, ground this pin
momentary to wake up

int sleepStatus = 0;

void setup() {

  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
// sets the digital pin as output

  pinMode(sleepPin, INPUT);
// sets the digital pin as input

  pinMode(wakePin, INPUT);

  Serial.println("Is this on?...");

}

void sleepNow(){

    digitalWrite(ledPin, LOW);


set_sleep_mode(SLEEP_MODE_PWR_DOWN)
;

    sleep_enable();

attachInterrupt(digitalPinToInterru
pt(wakePin),wakeUpNow, HIGH);

    sleep_mode();

    detachInterrupt(0);

}

void wakeUpNow(){

 digitalWrite(ledPin, HIGH);

  sleep_disable();

}

void loop() {

sleepStatus =
digitalRead(sleepPin);

  if (sleepStatus == LOW)

     sleepNow();

  else

  {

  if(Serial.available() > 0) {

    Serial.readBytes(r,63);

    Serial.print("Time: ");

    time = millis();

    Serial.println(time);

    Serial.println(r);

    }

   }

}
```

Lampiran 4

Program pada ESP *client*

```
#include <ESP8266WiFi.h>

const   char*   ssid   =   "Andika";
// Nama SSID AP/Hotspot

const     char*     password     =
"1234567890";      // Password Wifi

IPAddress        host(192,168,4,1);
// IP Server

char rcvd [30];

void setup() {

  Serial.begin(115200);

  delay(10);

// Connect to WiFi network -------
---------------------------------
-------

  Serial.println();

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

// Mengatur WiFi -----------------
---------------------------------
-------

  WiFi.mode(WIFI_STA);
// Mode Station

  WiFi.begin(ssid,       password);
// Mencocokan SSID dan Password

  while      (WiFi.status()      !=
WL_CONNECTED) {

    delay(100);

    Serial.print(".");

  }

// Print status Connect ----------
---------------------------------
-------

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

  Serial.println(WiFi.localIP());

    }

int value = 0;

void loop() {

 if(Serial.available() > 0) {

    Serial.readBytes(rcvd,18);

    String url = String (rcvd);

    delay(50);

    ++value;

    Serial.print("connecting  to
");

    Serial.println(host);

    // Use WiFiClient class to
create TCP connections

    WiFiClient client;

    const int httpPort = 80;

    if    (!client.connect(host,
httpPort)) {


Serial.println("connection
failed");

        return;

          }

    // This will send the request
to the server

    client.print(String("GET  ")
+ url + " HTTP/1.1\r\n\r\n");

    while (client.available() ==
0) {
```

```
            client.stop();
            return;
            }
      // Read all the lines of the
reply from server and print them
to Serial
     while(client.available()){
      String        line        =
client.readStringUntil('\r');
        Serial.print(line);
        }
        Serial.println();
        Serial.println("closing
connection");
    }
}
```

Lampiran 5

Program *Client* Arduino *Master*

```
#include <DHT.h>
#define DHTPIN 10
DHT dht(DHTPIN, DHT11,15);
char H [10];
char T [10];
int Relay = 7;
unsigned long waktu_2 = 0;
unsigned long waktu_1;
char* header = "#|01|0|";
void setup() {
  Serial.begin(115200);
  delay(5000);
  pinMode(Relay, OUTPUT);
  digitalWrite(Relay, LOW);
}
void loop() {
  waktu_1 = millis();
  boolean adadata = false ;
  if ( waktu_1 - waktu_2 >= 100){
      float h =dht.readHumidity();
      float                          t
=dht.readTemperature();
      if (isnan(h) || isnan(t))
       {
          Serial.println("Failed to
read from DHT sensor!");
          return;
       }
      // We now create a URI for the
request
String humidity    =(String(h));
String temperature =(String(t));
      humidity.toCharArray(H,5);
      temperature.toCharArray(T,5);
      adadata = true;
      waktu_2 = waktu_1;
    };
 if(adadata){
      Serial.write(header);
      Serial.write(H);
      Serial.write("|");
      Serial.write(T);
      Serial.write("|\n");
    };
}
}
```

Lampiran 6

Program *Client* Arduino *Slave*

```
#include <DHT.h>

#include <avr/sleep.h>

#define DHTPIN 10

DHT dht(DHTPIN, DHT11,15);

char H [10];

char T [10];

int Relay = 7;

unsigned long waktu_2 = 0;

unsigned long waktu_1;

char* header = "#|01|1|";

int ledPin = 13;                // LED
connected to digital pin 13

int sleepPin = 12;              // active
LOW, ground this pin momentary to
sleep

int wakePin = 2;                // active
LOW, ground this pin momentary to
wake up

int sleepStatus = 0;

void setup() {

  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);      //
sets the digital pin as output

  pinMode(sleepPin, INPUT);     //
sets the digital pin as input

  pinMode(wakePin, INPUT);

  delay (5000);

}

void sleepNow(){

    digitalWrite(ledPin, LOW);


set_sleep_mode(SLEEP_MODE_PWR_DOWN)
;

    sleep_enable();


attachInterrupt(digitalPinToInterru
pt(wakePin),wakeUpNow, HIGH);

    sleep_mode();

    detachInterrupt(0);

}

void wakeUpNow(){

  digitalWrite(ledPin, HIGH);

  sleep_disable();

}

void loop() {

  waktu_1 = millis();

  boolean adadata = false ;

  sleepStatus                       =
digitalRead(sleepPin);

  if (sleepStatus == LOW)

     sleepNow();

   else

   {

      if  (  waktu_1  -  waktu_2  >=
100){
```

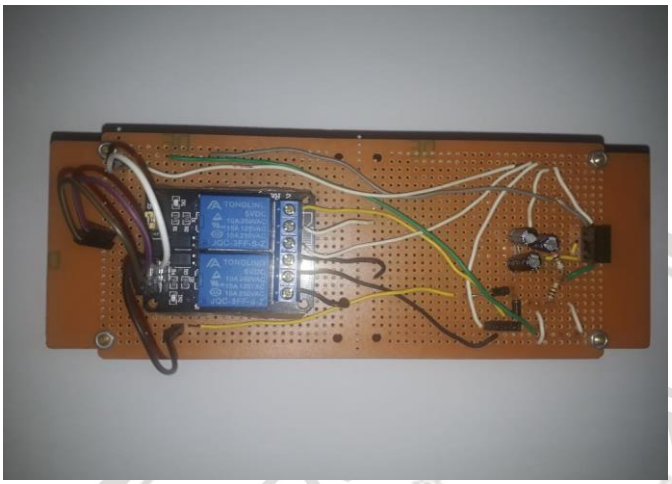
```
float h =dht.readHumidity();

float t =dht.readTemperature();

if (isnan(h) || isnan(t))

{

Serial.println("Failed to read from
DHT sensor!");

return;

     }

// We now create a URI for the
request

String humidity    =(String(h));

String temperature =(String(t));


humidity.toCharArray(H,5);
                                    temperature.toCharArray(T,5);

                                          adadata = true;

                                          waktu_2 = waktu_1;

                                         };

                                   if(adadata){

                                          Serial.write(header);

                                          Serial.write(H);

                                          Serial.write("|");

                                          Serial.write(T);

                                          Serial.write("|\n");

                                   };

                                    }

                                  }
```

Lampiran 7

Foto Alat dan Bahan yang Digunakan

A. Node *Client*

Tampak dari atas



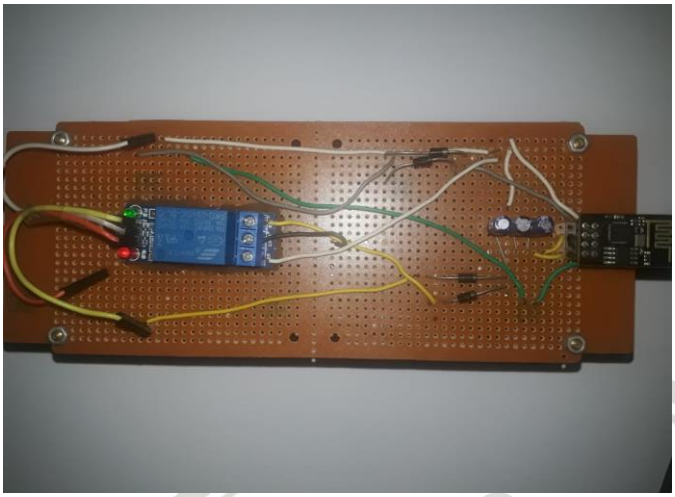
Tampak Sam*ping*

Bahan bahan yang diperlukan:

Arduino	: 2
ESP	: 1
Regulator 3,3 v	: 1
Kapasitor	: 1(0,1µF), 1 (10 µF), 1 (22µF)
Dioda	: 2
PCB dot matrix	: 2
Relay	: 2
Kabel jumper	

B. Node *Access Point*

Tampak dari atas



Tampak dari sam*pin*g

Bahan bahan yang diperlukan:

Arduino	: 2
ESP	: 1
Regulator 3,3 v	: 1
Kapasitor	: 1(0,1µF), 1 (10 µF), 1 (22µF)
Dioda	: 2
Relay	: 1
PCB dot matrix	: 2
Kabel jumper	

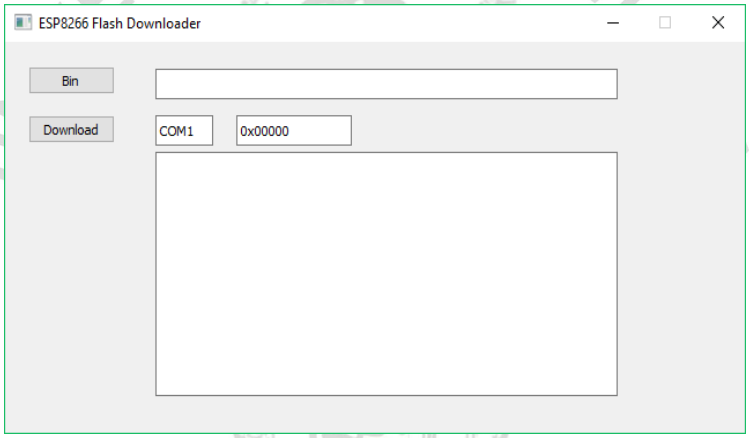
Lampiran 8

Flashing ESP8266-01

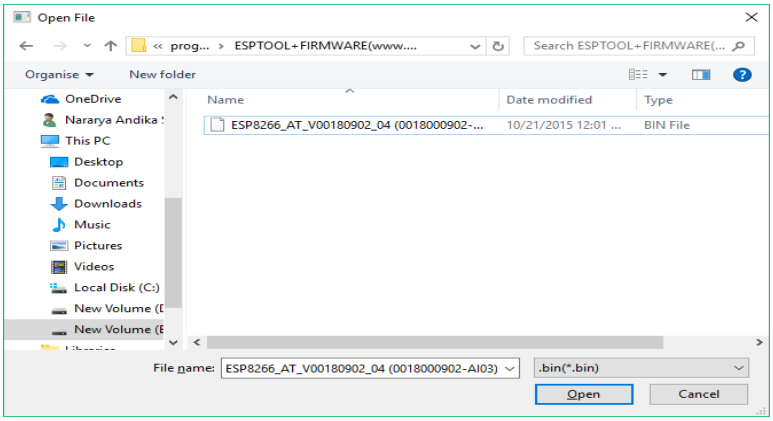
1. Menghubungkan *pin-pin* yang digunakan pada ESP ke arduino dan menkoneksikan ke komputer.

No	ESP	ARDUINO
1	Vin	3.3V
2	Ground	Ground
3	GPIO0	Ground
4	TX	TX
5	RX	RX
6	CH_PD	3.3V

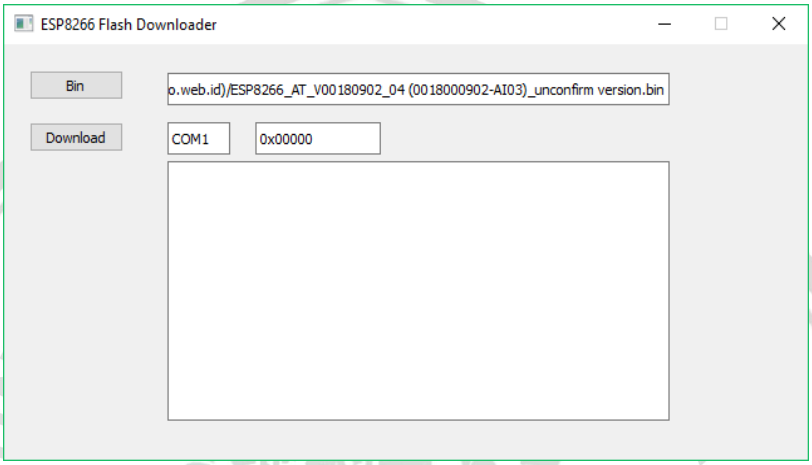
2. Membuka aplikasi ESPtool dan dan pilih pada bagian bin.



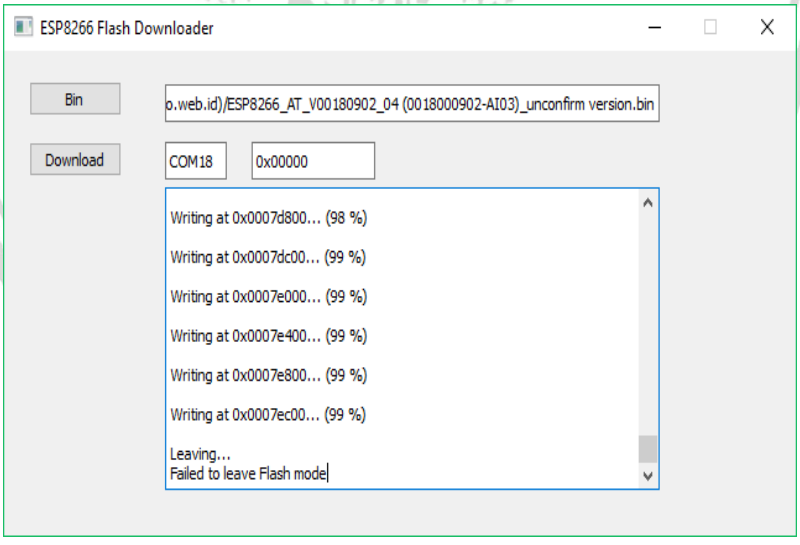
3. Setelah menakan tombol bin maka muncul *popup* windows dan pilih *firmware* ESP yang ber *ekstens*i (.bin) dan pilih open.



4. Setelah *firmware* dipilih maka pada port *serial* com pilih port *serial* yang digunakan.



5. Tekan tombol *download* dan *flashing* berjalan, tunggu hingga pemberitahuan (*leaving*) seperti gambar.



Lampiran 9

DATASHEET ATMEGA 328